

| Thu Jul 17 14:46:11 1997 | dpcagent.c | Page 1 |
|--|---|---|
| <pre> #include "dpcagent.h" /* Our header file */ /***** * History: * * Version 0.1 - First Package delivery for demo use. * Version 0.2 - (03-15-96) * * Site ID filtering to catalog * delivery support * * Stats() * * ig into file and added DpUpdateConfig() * Version 0.3 - (03-25-96) * * support for interactivity field in * * ee... in catalog by adding minimum * ore committing entry. * Version 0.4 - (03-29-96) * * indexes to modem configuration and * o 2400 to fix key reception bugs. * * to enable agent to continue getting * asful, even if agent shuts down. * Version 0.5 - (04-09-96) * * showing up in Cancel Download list * tatistics halfway running(doesn't abend) * explicit) files in catalogue * uebt/response/confirm code running but * it yet. * Version 0.6 - (04-10-96) * * nting Get MLID stats * alStrength() * Version 0.7 - (04-11-96) * * igation Editor * engh Meter * Version 0.8 - (04-12-96) * * alStrength to report 0 when not connected * nting cost(explicit) file download * View Database Entries option worked * entire entry and is more usable. * Version 0.9 - (04-16-96) *****/ </pre> | <pre> Internet support. Can't totally test away is up and running. * Version 0.10 - (04-24-96) * * ck * * lse" in IPSendRoutine after call * * ts. Turbo Internet works 90%. * Version 0.11 - (04-25-96) * * AppendStringField in DLO to include * * nstead of NULL. * Version 0.12 - (05-16-96) * * ase 2. * * between ATDT and 1-800... * * r commas in phone numbers and prefix * * ing to accept & and - * * Gateway strings to have (ISP) follow it * * played on if -DEBUG is on command line * Version 1.00 - (06-11-96) * * ud rate to Modem Status string * * it doesn't stay in during internet traffic * formation screen * * er edit strings to allow alphas * * buffer to 4K(default was 1K) * Version 1.01 - (06-17-96) * * iver comm code into driverio.c * * ster call to DPC.LAN so that it could * removes which would allow us to prevent * * Version 1.02 - (06-25-96) * * ch to disable package delivery for debug * Version 1.04 - (06-25-96) * * time to send directly to AIO. * Version 1.05 - (06-28-96) * * umber in menu to decimal * * size field to modem configuration * * which changed configuration screens * Version 1.06 - (07-05-96) * * te tinet fragment routines(jkt). * * rol menu. * * * Completed Turbo </pre> | <p>Page 2</p> <p>yet until IJ gat</p> <p>Added ARP loopba</p> <p>Fixed missing 'e</p> <p>to GatherFragmen</p> <p>Fixed NWSAppennd</p> <p>character sats i</p> <p>Completi of Ph</p> <p>Places x in</p> <p>Added support fo</p> <p>Allowed init str</p> <p>Modified IP and</p> <p>Debug screen dis</p> <p>Added connect ba</p> <p>Fixed Tx LED so</p> <p>Added Adapter in</p> <p>Fixed phone numb</p> <p>Bumped AIO write</p> <p>Broke out LAN dr</p> <p>Added App Regi</p> <p>call us it</p> <p>abends.</p> <p>Added -NOPD swit</p> <p>Revised send rou</p> <p>Changed serial n</p> <p>Added AIO buffer</p> <p>Added auto login</p> <p>Completely rewro</p> <p>Added Modem Cont</p> <p>Added GNU condit</p> |

```
ionals.
*
pt timeouts
*
utines into PPP.C
*
*****
#define AGENT_VERSION InxMSG("1.20 ", 173)
/*****
* Global variable used by DPCAGENT.C
*****
/
* Resource Tag variables.
*/
struct LoadDefinitionStructure *NLCHandle = 0;
struct ResourceTagStructure *allocrTag = 0;
struct ResourceTagStructure *timerTag = 0;
struct ResourceTagStructure *AESTag = 0;
struct ResourceTagStructure *asynctotag = 0;
/
* NUT and screen ID variables.
*/
NUTInfo
NUT handle
/* NUTHandle = NULL;
*/
/* main
*/
/* if DEBUG_ALL
*/
struct ScreenStruct *DebugScreenID = 0; /* for OutputToScreen
*/
#endef
WORD
/* Screen Height (25)
*/
WORD
/* Screen Width (80)
*/
LONG
ackground Portal
/*
* Process and thread variables.
*/
BYTE
inter to messages
LONG
DPCAgentPID = 0;
/* Main Handler thread PID
*/
LONG
DPCFilePID = 0;
/* Packet Handler thread PID
*/
LONG
DPCModemPID = 0;
/* Modem Handler thread PID
*/
LONG
DPCAccessPID = 0;
/* Access thread PID
*/
LONG
DPCInetPID = 0;
/* Turbo Internet thread PID
*/
BYTE
int
/* All threads must exit
*/
/* Tinet needs to wake up flag
*/
int
InetAsleep = FALSE;
/*
* CRC table used by calccrc().
*/
static unsigned short crcTab[256] = {
0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
0x8c48, 0x9dc1, 0xaf5a, 0xbcd3, 0xca6c, 0xdbef, 0xe97e, 0xf8f7,
0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,

```

```
0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xc87c, 0xd9f5, 0xf87c, 0xe9f5,
0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x66e2, 0x54b5, 0x453c,
0xbdc3, 0xac42, 0x9d99, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
0xcedc, 0xfdc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
0xdec3, 0xf444, 0xfdd3, 0xec56, 0x98e9, 0x8960, 0xbfbf, 0xaa72,
0x6306, 0x728f, 0x4014, 0x512d, 0x2522, 0x34ab, 0x0630, 0x17b9,
0xef4e, 0xfec7, 0xcc5c, 0xdd5d, 0xa96a, 0xb8e3, 0x8a78, 0x9bfb,
0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
0xfefc, 0xee46, 0xdcdd, 0xc5d4, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cfe,
0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0x93bb, 0xa232,
0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93bf,
0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
0x7b7c, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};
/* MLID Statistic globals */
int
GenericDescriptionTable[22*8] =
InxMSG("Total packets sent:", 103),
InxMSG("Total packets received:", 141),
InxMSG("No ECB available count:", 148),
InxMSG("Send packet too big count:", 156),
InxMSG("Reserved:", 186),
InxMSG("Receive packet overflow count:", 214),
InxMSG("Receive packet too big count:", 215),
InxMSG("Receive packet too small count:", 216),
InxMSG("Send packet miscellaneous errors:", 217),
InxMSG("Receive packet miscellaneous errors:", 218),
InxMSG("Send packet retry count:", 223),
InxMSG("Checksum errors:", 290),
InxMSG("Hardware receive mismatch count:", 291),
InxMSG("Total send OK byte count low:", 229),
InxMSG("Total send OK byte count high:", 230),
InxMSG("Total receive OK byte count low:", 231),
InxMSG("Total receive OK byte count high:", 232),
InxMSG("Total group address send count:", 233),
InxMSG("Total group address receive count:", 251),
InxMSG("Adapter reset count:", 252),
InxMSG("Adapter operating time stamp:", 253),
InxMSG("Adapter queue depth:", 255),
InxMSG("Send OK single collision count:", 159),
InxMSG("Send OK multiple collision count:", 256),
InxMSG("Send OK but deferred:", 264),
InxMSG("Send abort from late collision:", 265),
InxMSG("Send abort from excess collision:", 266),
InxMSG("Send abort from carrier sense:", 267),
InxMSG("Send abort from excessive deferral:", 268),
InxMSG("Receive abort from bad frame alignment:", 269)
```

```
);
#define STATS_DATA_WIDTH 13
#define FSD_DATA_WIDTH 40
#define BORDER_WIDTH 3
#define INDENT_WIDTH 3

void ( *BackgroundFuncPtr ) ( LONG portalNumber ) = NULL;
LONG BackgroundPortal;
int GenericLineStart;
int GblDataCol;
struct DOSCountryInfoStruct GblDOSCountryInfo;

int DebugFlag = FALSE;

/* The array nDeclination contains the Declination in degrees to add or
 * subtract from true azimuth to get magnetic azimuth.
 * The value 0 means that the location is not in mainland US
 */
static float nDeclination[] =
(
    0, 0, 4, 1, 0, 0, 0, 0, -8, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 1, -3, -4, -6, -8, -10, -11, 12, -13, 0, 0, 0,
    0, 0, 5, 2, -2, -5, -6, -9, -11, -12, -13, -13, -13, -14,
    0, 7, 5, 2, -1, -4, -6, -8, -12, -12, -14, -15, -16, -16,
    0, 12, 9, 6, -1, -4, -6, -9, -11, -12, -14, -16, -17, -17,
    18, 15, 0, 7, 2, -3, -6, -9, -12, -13, -15, -18, -19, -19,
    0, 0, 0, 0, 0, -1, -6, -9, -13, -15, -18, -20, -21, -22
);

#ifdef LOG_ECB_ACTIVITY
int DPC_TGID;
LogHandle LogClientHandle;
EventHandle LogECBHandle;
#endif /* LOG_ECB_ACTIVITY */

/*
 * Local function prototypes.
 */
void ReturnResources(int sig);
/*.....*/
ExitHandler(void *handle)

Description:
    This routine is called when the user hits Alt-F10 to exit
    the utility. We will attempt to verify with the user that
    they really want to exit.

Input:
    handle - NUT handle

Output:
    nothing

Returns:
    nothing
/*.....*/

void ExitHandler(void *handle)
(
    if (NWSConfirm(InxMSG("Exit DSEBUG?", 88), 0, 0, TRUE, NULL,
        handle, NULL) == TRUE)

```

```

    {
        ReturnResources(1);
        exit(1);
    }

static void NoSortHandler(List *head, LIST *tail, NUTInfo *handle)
(
    head = head;
    tail = tail;
    handle = handle;
    return;
)

/*.....*/
calcrc(WORD crc,
        BYTE *cp,
        LONG len)

Description:
    This routine calculates a CRC value for the text passed
    in. It uses a CRC substitution table for efficiency.
    Its used by the Slip transmit routine.

Input:
    crc
    RC.
    cp
    len
    Output:
    nothing

Returns:
    16-bit CRC value
/*.....*/
WORD calcrc(WORD crc, BYTE *cp, LONG len)
(
    while(len--)
        crc = (crc >> 8) ^ crctab[(crc ^ *cp++) & 0xff];
    return(crc);
)

/*.....*/
SlipSend(char *pdata,
          LONG sz,
          int cas,
          int timeout)

Description:
    This routine builds a SLIP envelope around the message
    passed in, escapes any HDLC characters in the message,
    and sends it to the modem.

Input:
    pdata
    to send
    sz
    message passed in
    - Pointer to the message
    - length of the

```

```

* send to hub
*
* timeout
*
* Output: nothing
*
* Returns: nothing
*
* .....
void SlipSend( char *pdata, LONG sz, int cas, int timeout)
(
    LROBPKT_t txcas;
    LROBPKT_t txhub;
    LONG tmlen;
    WORD crc;
    BYTE *pi, *po;
    BYTE slip_data[3072];
    int is, os;

    if (cas)
    (
        /* initialize cas transfer */
        MovB(pdata, txcas.data, sz);
        txcas.length = sz + (txcas.data - ((BYTE *)&txcas));
        tmlen = txcas.length;
        txcas.length |= 0x8000;
        crc = calcrc(INITCRC, (BYTE *)&txcas, tmlen);
        txcas.data[sz] = crc & 0xff;
        txcas.data[sz+1] = (crc >> 8) & 0xff;
        pi = (BYTE *) &txcas;

    )
    else
    (
        /* initialize hub transfer */
        MovB(pdata, txhub.data, sz);
        txhub.length = sz + (txhub.data - ((BYTE *)&txhub));
        txhub.filler1 = txhub.channel = 0;
        txhub.control = 0x8000;
        tmlen = txhub.length;
        crc = calcrc(INITCRC, (BYTE *)&txhub, tmlen);
        txhub.data[sz] = crc & 0xff;
        txhub.data[sz + 1] = (crc >> 8) & 0xff;
        pi = (BYTE *) &txhub;
    )

    /* Start out with SLIP start character */
    po = slip_data;
    *po++ = END;

    /* Escape any special SLIP characters that may be in the message */
    for (is = 0, os = 1; is < (tmlen + 2); is++, os++, pi++, po++)
    (
        switch(*pi)
        (
            case END:
                *po++ = ESC;
                *po = ESC_END;
                os++;
                break;
            case ESC:
                *po++ = ESC;
                *po = ESC_ESC;
                os++;
        )
    )
}

```

```

        break;
    default:
        *po = *pi;
        break;
    )
}

/* Finish packet off with SLIP END character */
*po = END;
os++;

/* Send message to the modem thread */
DioSend(slip_data, os, timeout);
}

.....
EXPORTED FUNCTION
DPCGetBoard(LONG *board,
             LONG *controlEntry)
{
    Description:
        This routine returns the DPC MLID logical board number
        and control entry address to be used to send IOCTL
        requests to the DPC mlid. The IOCTL mlid pragma can
        then be used to make the request.

    Input:
        board
        ore board number
        controlEntry - Pointer to where to store control entry
        y address

    Output:
        board and controlEntry filled in if successful

    Returns:
        0 if MLID is active
        .....
    LONG DPCGetBoard(LONG *board, LONG *controlEntry)
    (
        if (DIOBoard == 0)
            return(-1);

        *board = DIOBoard;
        *controlEntry = DIOControlEntry;
        return(0);
    )

    void
    CreateStringWithCommas( LONG number, BYTE *buffer, char *format )
    (
        int i;
        int j;
        int found;
        int length;
        int commasAdded = 0;
        BYTE tmpBuf[ 128 ];
        BYTE *tmpPtr;

        NWSprintf( tmpBuf, format, number);
        for ( i = ( length = CStrLen( tmpBuf ) ), found = 0; i >= 0; i-- )

```

```

    if ( ( tmpBuf[ i ] >= '0' ) && ( tmpBuf[ i ] <= '9' ) )
    {
        /* we have a digit */
        if ( ++found > 3 )
        {
            found = 1;

            /* shift the string one to the right */
            for ( j = ++length; j > i; j-- )
                tmpBuf[ j ] = tmpBuf[ j - 1 ];

            tmpBuf[ i + 1 ] = GblDOSCountryInfo.thousandSep;

            commasAdded++;
        }

        /* Adjust the length of the string back to its original if there are
        ** leading spaces.
        */
        for ( i = 0, tmpPtr = tmpBuf; i < commasAdded; i++ )
        {
            if ( *tmpPtr == ' ' )
                tmpPtr++;
        }
        CStrCpy( buffer, tmpPtr );
    }

void
SecondsToDateAndTime( LONG seconds, BYTE *buff )
{
    char *ascBuf;

    ascBuf = ascTime( localtime( (time_t *) &seconds ) );
    CMovB( ascBuf, buff, 26 );
    buff[ 24 ] = 0;

    void
    FormatElapsedTime( LONG seconds, LONG tenths, BYTE *buff )
    {
        LONG minutes, hours, days;

        /* convert secs to minutes */
        minutes = seconds / 60;
        seconds = seconds % 60;
        hours = minutes / 60;
        minutes = minutes % 60;
        days = hours / 24;
        hours = hours % 24;
        if ( days > 0 )
        {
            NWSprintf
            (
                buff,
                MSG( "%d%c%d%c%d%c%01d", 293 ),
                days,
                GblDOSCountryInfo.timeSep[ 0 ],
                hours,
                GblDOSCountryInfo.timeSep[ 0 ],
                minutes,
                GblDOSCountryInfo.timeSep[ 0 ],
                seconds,
                GblDOSCountryInfo.decimalSep[ 0 ],
                tenths
            );
        }
        else if ( minutes > 0 )
        {
            NWSprintf
            (
                buff,
                MSG( "%d%c%d%c%01d", 295 ),
                minutes,
                GblDOSCountryInfo.timeSep[ 0 ],
                seconds,
                GblDOSCountryInfo.decimalSep[ 0 ],
                tenths
            );
        }
        else
        {
            NWSprintf
            (
                buff,
                MSG( "%d%c%01d", 296 ),
                seconds,
                GblDOSCountryInfo.decimalSep[ 0 ],
                tenths
            );
        }
    }

    void HandleScrollablePortal( PCB *portal )
    {
        int escapeFlag = 0;
        LONG keyType;
        BYTE ch;
        int updatedDisplay = TRUE;
        LONG virtualHeight;
        LONG portalHeight;
        LONG bottomLine;
        LONG curPos;
        LONG vLine;

        /*
        ** The values for portal->portalHeight and portal->virtualHeight are the
        ** only two that we deal with that are 1-based. All the rest are 0-based
        */
    }
}

```

```

h
** so we will use local copies of these variables adjusted to fit in wit
** the rest in the PCB structure.
**/
virtualHeight = portal->virtualHeight - 1;
portalHeight = portal->portalHeight - 1;
/*
** Other initializations.
**/
bottomLine = virtualHeight - portalHeight;

while(!escapeFlag)
(
    if ( updatedDisplay == TRUE )
    /*
    ** The last iteration of the loop made a change, so redr
    ** portal.
    */
    if ( portal->verticalScroll == SCROLL_ON )
    (
        /*
        ** Adjust the vertical thumb on the right border
        */
        if ( portal->cursorLine == 0 )
        (
            if ( virtualHeight <= portalHeight )
                curPos = 100;
            else
                curPos = 0;
        )
        else
        (
            if ( portal->cursorLine >= bottomLine )
                curPos = 100;
            else
            (
                vLine = bottomLine;
                if ( vLine == 0 )
                    vLine = 1;

                curPos = ( portal->cursorLine *
                    100 ) / vLine;

                /* avoid divide by 0 */

                portal->showScrollBars &= ~VERTICAL_SCROLL_MASK;

                /*
                ** Display the vertical thumb at its new positio

```

```

OLL_SHIFT;
    portal->showScrollBars |= curPos << VERTICAL_SCR
)
NWSUpdatePortal( portal );
updatedDisplay = FALSE;
)
NWSGetKey( &keyType, &ch, NUTHandle );
switch ( keyType )
(
    case K_UP:
        /*
        ** Scroll the portal up one line.
        */
        if ( portal->virtualLine > 0 )
        (
            /*
            ** We're not at the top, so we can scrol
            */
            portal->virtualLine--;
            portal->cursorLine = portal->virtualLine
            updatedDisplay = TRUE;
        )
        break;

    case K_DOWN:
        /*
        ** Scroll the portal down one line.
        */
        if ( portal->virtualLine < bottomLine )
        (
            /*
            ** We're not at the bottom, so we can sc
            */
            portal->virtualLine++;
            portal->cursorLine = portal->virtualLine
            updatedDisplay = TRUE;
        )
        break;

    case K_PUP:
        /*
        ** Move the portal up one page.
        */
        if ( portal->cursorLine > 0 )
        (
            LONG    delta;

            /*
            ** We're not at the top, so we can move
            up. However, we need

```

```

** to figure out how much we can move up
*/
if ( portal->cursorLine > portalHeight )
    delta = portalHeight;
else
    delta = portal->cursorLine;

portal->cursorLine -= delta;
if ( portal->cursorLine < portal->virtua
rsorLine;

        updatedDisplay = TRUE;
    )

    break;

case K_DOWN:
    /* Move the portal down one page.
    */
    if ( portal->cursorLine < virtualHeight )
    (
        LONG delta;
        LONG newCurrentLine;
        /* We're not at the bottom, so we can mo
        ** we need to figure out how much we can
        */
        delta = virtualHeight - portal->cursorLi
        if ( delta > portalHeight )
            delta = portalHeight;
        newCurrentLine = portal->cursorLine + de
        delta = virtualHeight - portalHeight;
        if ( newCurrentLine > delta )
            portal->virtualline = delta;
        else
            portal->virtualline = newCurrent
        portal->cursorLine = portal->virtualline
        updatedDisplay = TRUE;
    )
    break;

case K_SUP:
    /* <Ctrl-PgUp> takes us to the top of the portal
    */
    portal->virtualline = 0;
    portal->cursorLine = 0;
    updatedDisplay = TRUE;
}

void UpdateStatsInformation(LONG portal)
(
    PCB *portalPtr;
    struct DriverStatsStructure *stats;

```

```

int line, count;
LONG
int numGenerics;
*statePtr;
string(80);
mask;
seconds;
tenths;
*customPtr;
*ptr;

NMSGGetPCB( &portalPtr, portal, NUTHandle );

if ( DPCGetMLIDStats(&stats))
{
    AddKey( NUTHandle->screenID, ENTER_KEY, 0, 0, 0 );
    return;
}

/* do generic stuff */
line = GenericLineStart;
statePtr = &( stats->NotSupportedMask );
numGenerics = stats->GenericVariableCount;
mask = *statePtr++;

for ( count = 0; count < numGenerics; count++ )
{
    if ( mask & ( 0x80000000 >> ( count & 0x1f ) ) )
    {
        /* not supported */
        NWSprintf( string, MSG("%13.13s", 301), MSG("Not support
ed", 298) );
        NWSShowPortalline
        (
            line++,
            GblDataCol,
            string,
            STATS_DATA_WIDTH,
            portalPtr
        );
        statePtr++;
    }
    else
    {
        if ( count == 20 )
        {
            BYTE tmpString( 80 );

            /*
            ** This is the operating time stamp, which needs
            ** output format.
            */
            ConvertTicksToSeconds( *statePtr++, &seconds, &t
            FormatElapsedTime( seconds, tenths, tmpString );
            NWSprintf( string, MSG("%13.13s", 302), tmpStrin
            );
            else
            {
                CreateStringWithCommas
                (
                    *statePtr++,
                    string,
                    MSG("%13lu", 303)
                );
                NWSShowPortalline
                (
                    line++,
                    GblDataCol,
                    string,
                    STATS_DATA_WIDTH,
                    portalPtr
                );
            }
        }
        /* do custom stuff */
        line += 2;
        customPtr = (CustVars *)(&stats->CustomVariableCount);
        ptr = (BYTE *)(&customPtr->CustomVariable(customPtr->CustomVariableCount)
        );
        ptr += sizeof(WORD);
        for ( count = 0; count < customPtr->CustomVariableCount; count++ )
        {
            CreateStringWithCommas
            (
                customPtr->CustomVariable( count ),
                string,
                MSG("%13lu", 299)
            );
            NWSShowPortalline( line++, GblDataCol, string, STATS_DATA_WIDTH,
            portalPtr );
        }
        NWSUpdatePortal( portalPtr );
    }
}

void SignalMeter(void) {
    int exitNow = FALSE;
    LONG type;
    BYTE value;
    int signal = 0;
    int oldsignal = 0;
    int avgSqf = 0;
    int beamSize;
    int beamPercent;
    int block = FALSE;
    int rxFreq;
    struct DriverStatsStructure *stats;
    CustVars
        *customPtr;
    char freqStr(80);
    char aveStr(80);
    char signalStr(80);
    int sound_gap = 0;
    while(!exitNow) {
        if (DPCGetMLIDStats(&stats)) {
            type = signal = 0;
            rxFreq = 0;
        }
        else {
            customPtr = (CustVars *)(&stats->CustomVariableCount);
            type = signal = customPtr->CustomVariable(0);
        }
    }
}

```

```

    rxFreq = customPtr->CustomVariable[1];
}

NWSprintf(freqStr, MSG(" Frequency : %d", 345), rxFreq / 10);

if (signal >= 200)
    signal = (2 * (signal - 200)) + 60;
else
    signal = 0;

if (avgSqf == 0L)
    avgSqf = 100L * signal;
avgSqf = ((avgSqf * 19L) + (100L * (unsigned long) signal)) / 20L;

beamSize = (int)((avgSqf/100L - 60L)/2L);
beamSize *= 5;
beamSize /= 4;

if (beamSize < 0)
    beamSize = 0;
else if (beamSize >= MAX_BEAM)
    beamSize = MAX_BEAM - 1;

beamPercent = (100 * beamSize) / MAX_BEAM;

if (oldSignal != signal) {
    if (signal < MIN_SQF_VAL)
        block = FALSE;
    else
        block = TRUE;
    oldSignal = signal;
}

NWSprintf(aveStr, MSG(" Average SQF : %d", 346),
    signal ? avgSqf / 100L : 0);

NWSprintf(signalStr, MSG("Signal Quality : %d (%d%%)",
    signal, beamPercent),
    block ? MSG("Signal Locked", 348) : MSG("Signal Not Locked", 349));

DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 341),
    beamSize,
    MAX_BEAM,
    freqStr,
    aveStr,
    signalStr);

if (--sound_gap <= 0) {
    /* calculate frequency based on raw signal strength */
    signal = 200 + type;
    /* adjust for 8253-5 timer chip output */
    signal = 1193180 / signal;

    /* turn on sound */
    outp(67, 182);
    outp(66, signal % 256);
    outp(66, signal / 256);
    outp(97, inp(97) | 0x03);

    delay(300);

    /* turn off sound */
    outp(97, inp(97) & ~0x03);
}

rx_sound_gap = (110 - beamPercent) / 17;

delay(150);

if (NWSKeyStatus(NUTHandle)) {
    NWSGetKey(&type, &value, NUTHandle);
    if ((type == K_ESCAPE) || (type == K_AF10))
        exitNow = TRUE;
}

/* Destroy the throttle portal */
DisplayThrottle(MSG("Satellite Dish Signal Strength Meter", 351),
    MAX_BEAM,
    freqStr,
    aveStr,
    signalStr);
}

void GetRegionName(char *regionName)
{
    FILE *fp;
    char szTmp[128];
    char *src, *dest;
    LONG len;

    fp = fopen(MSG("country.ini", 635), MSG("r", 636));
    len = strlen(MSG("RegionName=", 637));
    *regionName = 0;
    if (fp != NULL)
    {
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strnicmp(szTmp, MSG("RegionName=", 638), len)) ==
                0 )
            {
                fclose(fp);
                src = &szTmp[len];
                dest = regionName;
                while(*src != 0 && *src != ' ' && *src != '\n')
                {
                    *dest = *src;
                    src++;
                    dest++;
                }
                *dest = 0;
                return;
            }
        }
        fclose(fp);
    }

    void GetCountry(char *countryName)
    {
        FILE *fp;
        char szTmp[128];
        char *src, *dest;
        LONG len;
    }
}

```

```

fp = fopen(MSG("country.ini", 639), MSG("r", 640));
len = strlen(MSG("Name=", 641));
*countryName = 0;
if (fp != NULL)
{
    while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
    {
        if ( (strnicmp(szTmp, MSG("Name=", 642), len)) == 0)
        {
            fclose(fp);
            src = &szTmp[len];
            dest = countryName;
            while(*src != 0 && *src != ' ' && *src != '\r' &
                (
                    *dest = *src;
                    src++;
                    dest++;
                )
            {
                *dest = 0;
                return;
            }
            fclose(fp);
        }
    }
}

void GetDefaultService(char *serviceName)
{
    FILE *fp;
    char szTmp[128];
    char *src, *dest;
    LONG len;

    fp = fopen(MSG("country.ini", 626), MSG("r", 627));
    len = strlen(MSG("Service=", 628));
    *serviceName = 0;
    if (fp != NULL)
    {
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        {
            if ( (strnicmp(szTmp, MSG("Service=", 185), len)) == 0)
            {
                fclose(fp);
                src = &szTmp[len];
                dest = serviceName;
                while(*src != 0 && *src != ' ' && *src != '\r' &
                    (
                        *dest = *src;
                        src++;
                        dest++;
                    )
                {
                    *dest = 0;
                    return;
                }
            }
            fclose(fp);
        }
    }
}

typedef struct
{

```

```

char name[20];
LONG longitude;
LONG eastFlag;
LONG frequency;
LONG horzFlag;
LONG cityLongDegrees;
LONG cityLongMinutes;
LONG cityLatDegrees;
LONG cityLatMinutes;
LONG cityEastFlag;
LONG cityNorthFlag;
) SatelliteInfo;

typedef struct
{
    float elevation;
    float trueAzimuth;
    float magAzimuth;
    float polarization;
} DishInfo;

```

```

void ParseSatelliteInfo(char *satName, int nameContainsInfo, SatelliteInfo *sat)
{
    FILE *fp;
    char szTmp[128];
    char *fptr;
    char *name, *cptr;
    char *ascPtr;
    char ascBuf[10];

    if (nameContainsInfo == FALSE)
    {
        fp = fopen(MSG("country.ini", 644), MSG("r", 629));
        if (fp != NULL)
        {
            while ((fptr = fgets(szTmp, sizeof(szTmp) - 1, fp)) != N
                (
                    if ( (strnicmp(szTmp, satName, strlen(satName)))
                        break;
                )
            {
                if (fp) fclose(fp);
                if (!fptr)
                    return;
                cptr = szTmp;
            }
            else
                cptr = satName;

            name = sat->name;
            while(*cptr != '\0')
                *name++ = *cptr++;
            *name = 0;

            cptr++;
            while(*cptr == ' ')
                cptr++;

            ascPtr = ascBuf;
            while(*cptr != ',')
                *ascPtr++ = *cptr++;
        }
    }
}

```

```

    cptr++;
    *ascPtr = 0;
    sat->longitude = atoi(ascBuf);

    if (*cptr == 'e' || *cptr == 'E')
        sat->eastFlag = 1;
    else
        sat->eastFlag = 0;

    while(*cptr != ',')
        cptr++;

    cptr++;
    ascPtr = ascBuf;
    while(*cptr != ',')
        *ascPtr++ = *cptr++;
    cptr++;
    *ascPtr = 0;
    sat->frequency = atoi(ascBuf);

    if (*cptr == 'h' || *cptr == 'H')
        sat->horzFlag = 1;
    else
        sat->horzFlag = 0;

    )

void GetDefaultSatellite(SatelliteInfo *sat)
(
    FILE *fp;
    char szTmp[128];
    LONG len;
    char *ccode;

    fp = fopen(MSG("country.ini", 632), MSG("r", 633));
    len = strlen(MSG("Hughes Networks", 630));
    if (fp != NULL)
    (
        while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
        (
            if ( (strnicmp(szTmp, MSG("Hughes Networks"), 631), len
            )) == 0)
            (
                FILE *fp;
                char szTmp[128];
                LONG len;
                char *ccode;

                fp = fopen(MSG("country.ini", 632), MSG("r", 633));
                len = strlen(MSG("Hughes Networks", 630));
                if (fp != NULL)
                (
                    while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                    (
                        ccode = fgets(szTmp, sizeof(szTmp) - 1, fp);
                        fclose(fp);
                        if (ccode == NULL)
                            return;

                        ParseSatelliteInfo(szTmp, TRUE, sat);
                        return;
                    )
                    fclose(fp);
                )
            )

        int NewCalculationFlag = 0;

        LONG ChangeSatellite(FIELD *fieldPtr, int key, int *changed, NUTInfo *handle)
        (
            FILE *fp;
            SatelliteInfo *sat;
            LIST *listPtr = NULL;
            LONG ccode;
            LONG rcode = K_SELECT;

            ParseSatelliteInfo(listPtr->text, FALSE, sat);

```

```

        char *fstr;
        char szTmp[128];
        char *szPtr;
        int rows = 0, cols = 0, len;
        void (*oldSortFunction) (LIST *, LIST *, NUTInfo *);
        key = key;
        changed = changed;
        handle = handle;

        sat = (SatelliteInfo *)fieldPtr->customData;

        NWSGetListSortFunction(NUTHandle, &oldSortFunction);
        NWSSetListSortFunction(NUTHandle, NoSortHandler);

        if (NWSPushList(NUTHandle) == 0)
        (
            NWSSetListSortFunction(NUTHandle, oldSortFunction);
            return rcode;
        )

        NWSInitList(NUTHandle, NULL);

        fp = fopen(MSG("country.ini", 634), MSG("r", 667));
        len = strlen(MSG("Hughes Networks", 668));
        if (fp != NULL)
        (
            while ((fstr = fgets(szTmp, sizeof(szTmp) - 1, fp)) != NULL)
            (
                if ( (strnicmp(szTmp, MSG("Hughes Networks"), 205), len
                )) == 0)
                (
                    break;
                )
            )
            if (fstr != NULL)
            (
                while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
                (
                    if (*szTmp == ' ' || *szTmp == '\r' || *szTmp == '\n')
                        break;
                    szPtr = szTmp;
                    while (*szPtr != '=')
                        szPtr++;
                    *szPtr = 0;
                    AppendToList(szTmp, 0, &rows, &cols);
                )
            )

            if (rows == 0)
                goto ChangeSatExit;

            ccode = NWSList(
                InxMSG("Choose Satellite", 648),
                12, 40,
                (rows > 16) ? 16 : rows,
                (cols < 20) ? 20 : cols,
                M_ESCAPE | M_SELECT,
                &listPtr,
                NUTHandle, NULL,
                NULL, NULL);

            if (ccode == M_SELECT)
            (
                ParseSatelliteInfo(listPtr->text, FALSE, sat);

```

```

    NewCalculationFlag = 1;
    rcode = K_ESCAPE;
}

ChangeSatExit:
    NWSdestroyList(NUTHandle);
    NWSPopList(NUTHandle);
    NWSSetListSortFunction(NUTHandle, oldSortFunction);
    return rcode;
}

LONG
ComputeHotSpot(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
    fp = fp;
    key = key;
    changed = changed;
    handle = handle;
    NewCalculationFlag = 1;
    return K_ESCAPE;
}

int
LongitudeHemisphereHandler(int option, void *parameter)
{
    parameter = parameter;
    return option;
}

int
PolarizationHandler(int option, void *parameter)
{
    parameter = parameter;
    return option;
}

int
GroundLatHemHandler(int option, void *parameter)
{
    parameter = parameter;
    return option;
}

int
GroundLongHemHandler(int option, void *parameter)
{
    parameter = parameter;
    return option;
}

float SGN(float num)
{
    if (num < 0)
        return -1;
    return 1;
}

void CalculateDish(SatelliteInfo *sat, DishInfo *dish)
{
    float LW, LN, ZR, YR, XR, DIST, EL, PO, AZ, TRAZ, S, A;
    float SD, RD, RM, LD, LM;
    static const float M_PI = 3.14159;
    static const float RAD = 6378.388;
    static const float ALT = 42164.24;
    int count = 4;

    SD = p->dSatLong;
    RD = p->dRemLongDeg;
    RM = p->dRemLongMin;

```

```

    LD = p->dRemLatDeg;
    LM = p->dRemLatMin;

    SD = (float)sat->longitude;
    RD = (float)sat->cityLongDegrees;
    RM = (float)sat->cityLongMinutes;
    LD = (float)sat->cityLatDegrees;
    LM = (float)sat->cityLatMinutes;

    SD = fabs(SD);
    if (p->cSatLong == 'E')
        if (sat->eastFlag)
            SD = -SD;

    RD = fabs(RD);
    RM = fabs(RM);
    if (p->cRemLong == 'E')
        if (sat->cityEastFlag)
        {
            RD = -RD;
            RM = -RM;
        }

    LD = fabs(LD);
    LM = fabs(LM);

    if (p->cRemHem == 'S')
        if (sat->cityNorthFlag == 0)
        {
            LD = -LD;
            LM = -LM;
        }

    LW = ((RD + RM / 60) - SD) * M_PI / 180;
    if (LW == 0)
        LW = .00001;
    LN = (LD + LM / 60) * M_PI / 180;
    if (LN == 0)
        LN = .00001;

    ZR = RAD * sin(LN);
    YR = RAD * cos(LN) * cos(LW);
    XR = RAD * cos(LN) * sin(LW);
    DIST = sqrt(XR * XR + (ALT - YR) * (ALT - YR) + ZR * ZR);
    EL = (ALT * YR - RAD * RAD) / (RAD * DIST);
    EL = atan(EL / sqrt(1 - EL * EL)) * 180 / M_PI;
    EL = (10 * EL + .5 * SGN(EL)) / 10;
    A = 0;
    if (LN * LW > 0)
        A = 2 * M_PI;
    if (LN > 0)
        A = M_PI;

    AZ = (A - atan(tan(LW) / sin(LN))) * 180 / M_PI;
    AZ = floor(10 * AZ + .5 * SGN(AZ)) / 10;
    TRAZ = AZ;

    /* Executed for US Mainland only */
    /* Declination correction to TRUE Azimuth */
    if (((LD > 23) && (LD < 50)) && ((RD > 70) && (RD < 125)))

```

```

(
    LD = floor(LD/4) - 6;
    RD = ceil(RD/4) - 18;
    if((inDeclination((int)(LD*14+RD)))
        count--;
    if((inDeclination((int)(LD*14+RD-1)))
        count--;
    if((inDeclination((int)((LD+1)*14+RD-1)))
        count--;
    if((inDeclination((int)((LD+1)*14+RD)))
        count--;
    if(count)
    {
        AZ = AZ*(inDeclination((int)(LD*14+RD)))
        +nDeclination((int)(LD*14+RD-1))
        +nDeclination((int)((LD+1)*14+RD))
        +nDeclination((int)((LD+1)*14+RD-1))/count;
    }

    if (AZ >= 360) AZ = AZ - 360;
    if (AZ < 0) AZ = AZ + 360;

    S = tan(LN) / sin(LW);

    PO = atan(S);
    /* printf("S=%f LN=%f LW=%f PO=%f\n",S,LN,LW,PO); */

    PO = fabs(PO);
    PO = M_PI / 2.0 - PO;
    PO = PO * SGN(S) * 180 / M_PI;
    PO = floor(10 * PO + .5 * SGN(PO)) / 10;

    // p->dRemElev = EL;
    // p->dRemMagAz = AZ;
    // p->dRemTrueAz = TRAZ;
    // p->dRemPolar = -PO;
    dish->elevation = EL;
    dish->magAzimuth = AZ;
    dish->>trueAzimuth = TRAZ;
    dish->polarization = -PO;
)

void ComputeCity(char *region, char *city, LONG latLong)
{
    FIELD *fp;
    char country[20], countryStr[30];
    char regionStr[30], cityStr[30];
    char service[30], serviceStr[30];
    char satelliteStr[50];
    char elevationStr[50], trueAzimuthStr[50];
    char magAzimuthStr[50], polarizationStr[50];
    SatelliteInfo sat;
    DishInfo dish;
    int i;
    int start;
    MFCNTROL *mfct10, *mfct11, *mfct12, *mfct13;

    calculateAgain;
    NWSInitForm(NUTHandle);
    if (NewCalculationFlag == 0)
    {

```

```

        sat.cityLatDegrees = (latLong >> 24) & 0xff;
        sat.cityLatMinutes = (latLong >> 16) & 0xff;
        sat.cityLongDegrees = (latLong >> 8) & 0xff;
        sat.cityLongMinutes = latLong & 0xff;
        GetDefaultSatellite(&sat);
        sat.cityEastFlag = sat.eastFlag;
        sat.cityNorthFlag = 1;
    }
    NewCalculationFlag = 0;

    i = 0;
    GetCountry(country);
    NWSprintf(countryStr, MSG("Country : %s", 488), country);
    NWSAppendCommentField(i, 2, countryStr, NUTHandle);

    NWSprintf(regionStr, MSG("Region : %s", 712), region);
    NWSAppendCommentField(i, 36, regionStr, NUTHandle);

    i++;
    NWSprintf(cityStr, MSG("City : %s", 713), city);
    NWSAppendCommentField(i, 2, cityStr, NUTHandle);

    GetDefaultService(service);
    NWSprintf(serviceStr, MSG("Service : %s", 714), service);
    NWSAppendCommentField(i, 36, serviceStr, NUTHandle);

    i++;
    NWSprintf(satelliteStr, MSG("Satellite : %s", 649), sat.name);
    start = (78 - strlen(satelliteStr)) / 2;
    fp = NWSAppendHotSpotField(i, start, NORMAL_FIELD, satelliteStr,
        ChangesSatellite, NUTHandle);

    fp->customData = &sat;

    i++;
    NWSAppendCommentField(i, 2, MSG("Satellite Longitude : ", 715), NUTHa
        ndle);
    NWSAppendIntegerField(i, 27, NORMAL_FIELD, (int *)&sat.longitude, 1, 180
        , F_NO_HELP, NUTHandle);

    NWSAppendCommentField(i, 34, MSG("Hemisphere : ", 716), NUTHandle);
    mfct10 = NWSInitMenuField(InxMSG("Satellite Longitude Hemisphere", 717),
        10, 40, LongitudeHemisphereHandler, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("West", 718), 0, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("East", 719), 1, NUTHandle);
    NWSAppendMenuField(i, 47, NORMAL_FIELD, (int *)&sat.eastFlag, mfc
        LL, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Satellite Polarization : ", 489), NUTHa
        ndle);
    mfct11 = NWSInitMenuField(InxMSG("Satellite Polarization", 490), 10, 40,
        PolarizationHandler, NUTHandle);
    NWSAppendToMenuField(mfct11, InxMSG("Vert", 531), 0, NUTHandle);
    NWSAppendToMenuField(mfct11, InxMSG("Horz", 539), 1, NUTHandle);
    NWSAppendMenuField(i, 27, NORMAL_FIELD, (int *)&sat.horzFlag, mfc
        LL, NUTHandle);

    NWSAppendCommentField(i, 34, MSG("Frequency : ", 541), NUTHandle);
    NWSAppendIntegerField(i, 47, NORMAL_FIELD, (int *)&sat.frequency, 1, 100
        00, F_NO_HELP, NUTHandle);

    i+=2;
    NWSAppendCommentField(i, 2, MSG("Ground Longitude degrees : ", 543), NUT
        Handle);
    NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&sat.cityLongDegrees,
        1, 180, F_NO_HELP, NUTHandle);

```

```

NWSAppendCommentField(i, 40, MSG("minutes : ", 545), NUTHandle);
NWSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)&sat.cityLongMinutes,
1, 180, F_NO_HELP, NUTHandle);

NWSAppendCommentField(i, 58, MSG("Hemisphere : ", 686), NUTHandle);
mfct12 = NWSInitMenuField(InxMSG("Ground Longitude Hemisphere", 687), 10
40, GroundLongHandler, NUTHandle);
NWSAppendToMenuField(mfct12, InxMSG("West", 688), 0, NUTHandle);
NWSAppendToMenuField(mfct12, InxMSG("East", 689), 1, NUTHandle);
NWSAppendMenuField(i, 71, NORMAL_FIELD, (int *)&sat.cityEastFlag, mfct12
, NULL, NUTHandle);

i++;
NWSAppendCommentField(i, 2, MSG("Ground Latitude degrees : ", 690), NUT
Handle);
NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&sat.cityLatDegrees, 1
, 180, F_NO_HELP, NUTHandle);

NWSAppendCommentField(i, 40, MSG("minutes : ", 691), NUTHandle);
NWSAppendIntegerField(i, 50, NORMAL_FIELD, (int *)&sat.cityLatMinutes, 1
, 180, F_NO_HELP, NUTHandle);

NWSAppendCommentField(i, 58, MSG("Hemisphere : ", 692), NUTHandle);
mfct13 = NWSInitMenuField(InxMSG("Ground Latitude Hemisphere", 693), 10,
40, GroundLatHandler, NUTHandle);
NWSAppendToMenuField(mfct13, InxMSG("South", 694), 0, NUTHandle);
NWSAppendToMenuField(mfct13, InxMSG("North", 695), 1, NUTHandle);
NWSAppendMenuField(i, 71, NORMAL_FIELD, (int *)&sat.cityNorthFlag, mfct1
3, NULL, NUTHandle);

i++;
NWSAppendHotSpotField(i, 35, NORMAL_FIELD, MSG("COMPUTE NOW", 696),
ComputeHotSpot, NUTHandle);

CalculatedDish(&sat, &dish);

if (sat.horzFlag == 0)
{
    dish.polarization += 90.0;
    if (dish.polarization > 90.0)
        dish.polarization -= 90.0;
    if (dish.polarization < -90.0)
        dish.polarization += 90.0;
}

i++;
NWSprintf(elevationStr, MSG(" Elevation : %lf", 697), dish.elev
ation);
NWSprintf(trueAzimuthStr, MSG(" True Azimuth : %lf", 698), dish.true
Azimuth);
// if ((strcmp(country, MSG("USA", 243))) != 0)
//     strcpy(magAzimuthStr, MSG("Magnetic Azimuth : N/A", 699));
// else
//     NWSprintf(magAzimuthStr, MSG("Magnetic Azimuth : %lf", 568), d
ish.magAzimuth);
NWSprintf(polarizationStr, MSG(" Polarization : %lf", 700), dish.pola
rization);

NWSAppendCommentField(i, 2, elevationStr, NUTHandle);
i++;
NWSAppendCommentField(i, 2, trueAzimuthStr, NUTHandle);
i++;
NWSAppendCommentField(i, 2, magAzimuthStr, NUTHandle);
i++;

NWSAppendCommentField(i, 2, polarizationStr, NUTHandle);
i++;

NWSEditPortalForm(InxMSG("Antenna Pointing Calculations", 701),
12, 40,
/* center line, column */
/* form height, width */
F_NOVERIFY, F_NO_HELP,
/* Control flags, help m
essage */
NULL,
NUTHandle);

/* Confirm message, hand
le */

NWSDestroyForm(NUTHandle);
if (NewCalculationFlag != 0)
    goto calculateAgain;
}

void ComputeGetCity(char *region)
{
    DIR *dirCountry, *dirCity;
    char *name;
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    int rows = 0, cols = 0;
    FILE *fp;
    char szTmp[128];
    char byteStr[8];
    int latDeg, latMin, longDeg, longMin;
    int len, start = 0;
    LONG info;

    NWSStartWait( 0, 0, NUTHandle ); /* DMH change 970131 */

    getCitiesLoop:
        rows = cols = 0;
        listPtr = NULL;
        NWSInitList(NUTHandle, NULL);
        dirCountry = opendir(MSG("*.cty", 702));
        if (dirCountry == NULL)
            goto errorNoDir;

        while( (dirCity = readdir(dirCountry)) != NULL )
        {
            name = dirCity->d_name;
            fp = fopen(name, MSG("r", 703));
            if (fp == NULL)
            {
                closedir(dirCountry);
                goto errorNoDir;
            }
            len = strlen(region);
            if (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
            {
                if ( (strcmp(region, szTmp, len)) == 0 )
                    break;
            }
            fclose(fp);
        }
}

```

```

if (dirCity == NULL)
{
    closedir(dirCountry);
    goto errorNoDir;
}

while (fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
{
    int len = strlen(szTmp) - 2;
    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    byteStr[3] = 0;
    longMin = atoi(&byteStr[1]);

    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    byteStr[0] = szTmp[len--];
    longDeg = atoi(&byteStr);

    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    latMin = atoi(&byteStr[1]);

    byteStr[2] = szTmp[len--];
    byteStr[1] = szTmp[len--];
    latDeg = atoi(&byteStr[1]);

    if (longMin > 59)
    {
        longDeg++;
        longMin = 0;
    }

    if (latMin > 59)
    {
        latDeg++;
        latMin = 0;
    }

    info = ((latDeg & 0xff) << 24) |
            ((latMin & 0xff) << 16) |
            ((longDeg & 0xff) << 8) |
            (longMin & 0xff);

    while(szTmp[len] == ' ')
        len--;
    if (len > 0)
    {
        szTmp[len+1] = 0;
        while(len)
        {
            if (szTmp[len] == ' ')
                start = len + 1;
            len--;
        }
        if (start > 0)
        {
            AppendToList(&szTmp[start], info, &rows, &cols);
        }
    }
    fclose(fp);
    if (rows == 0)
    {
        errorNoDir:
        NWSWait( NUTHandle ); /* DMH change 970131 */
        NWSDestroyList(NUTHandle);
        NWSAlert(12, 40, NUTHandle, INXMSG("Unable to access Cities file", 705));
        return;
    }

    void ComputedGetRegion(char *country)
    {
        DIR *dirCountry, *dirCity;
        char *name, *end;
        LIST *listPtr = NULL;
        LONG ccode = M_SELECT;
        int rows = 0, cols = 0;
        char path[64];
        FILE *fp;
        char szTmp[128];
        LONG header;

        NWSWait( 0, 0, NUTHandle ); /* DMH change 970131 */
        getRegionsLoop:
        rows = cols = 0;
        listPtr = NULL;
        NWSInitList(NUTHandle, NULL);
        SetCurrentNameSpace(DOSNameSpace);
        NWSprintf(path, MSG("SYS:DIRECPC\\DB\\%s.cou", 706), country);
        if (chdir(path))
            goto errorNoDir;

        dirCountry = opendir(MSG("..cty", 707));
        if (dirCountry == NULL)
            goto errorNoDir;
    }
}

```

```

while( (dirCity = readdir(dirCountry)) != NULL )
{
    name = dirCity->d_name;
    fp = fopen(name, MSG("r", 708));
    if (fp == NULL)
    {
        closedir(dirCountry);
        goto errorNoDir;
    }
    if(fgets(szTmp, sizeof(szTmp) - 1, fp) != NULL)
    {
        end = &szTmp[strlen(szTmp) - 2];
        while( (*end == ' ') && (end != szTmp) )
            end--;
        end++;
        *end = 0;
        AppendToList(szTmp, 0, &rows, &cols);
    }
    fclose(fp);
}

if (rows == 0)
{
    closedir(dirCountry);
    goto errorNoDir;
}

GetRegionName(szTmp);
if ( (strcmpi(szTmp, MSG("Province", 709))) == 0)
    header = InxMSG("Choose A Province", 710);
else
    header = InxMSG("Choose A State", 711);

if (rows == 1)
{
    NWSendWait( NUTHandle );
    NWSDestroyList(NUTHandle);
    ComputeGetCity(szTmp);
    closedir(dirCountry);
    return;
}
else
{
    NWSendWait( NUTHandle );
    ccode = NWSList(
        header,
        12, 40,
        (rows < 16) ? rows : 16,
        (cols < 18) ? 18 : cols,
        M_ESCAPE | M_SELECT,
        &listPtr,
        NUTHandle, NULL,
        NULL, NULL);

    /* Height */
    /* Width */

    strcpy(szTmp, listPtr->text);
    NWSDestroyList(NUTHandle);
    closedir(dirCountry);
    ComputeGetCity(szTmp);
    goto getRegionsLoop;
}

if (ccode == M_SELECT)
{

```

```

NWSDestroyList(NUTHandle);

```

```

closedir(dirCountry);
return;

```

```

errorNoDir:

```

```

    NWSendWait( NUTHandle );
    NWSDestroyList(NUTHandle);
    NWSAlert(12, 40, NUTHandle, InxMSG("Unable to locate Country files in Co
untry directory %s.cou", 569), country);
    return;
}

```

```

void ComputeCoordinates(void)
{

```

```

    DIR *dirDB, *dirCountry;
    char *name, *dot;
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    int rows = 0, cols = 0;

```

```

    getCountriesLoop:

```

```

        rows = cols = 0;
        listPtr = NULL;
        NWSInitList(NUTHandle, NULL);
        SetCurrentNameSpace(DOSNameSpace);
        if (chdir(MSG("-SYS:DIREPC\\DB", 570)))
            goto errorNoDir;

```

```

        dirDB = opendir(MSG("**.cou", 571));
        if (dirDB == NULL)
            goto errorNoDir;

```

```

        while( (dirCountry = readdir(dirDB)) != NULL )
        {

```

```

            name = dirCountry->d_name;
            if ((dot = strchr(name, '.')) != NULL)
                *dot = 0;
            AppendToList(name, 0, &rows, &cols);
        }

```

```

        if (rows == 0)
        {

```

```

            closedir(dirDB);
            goto errorNoDir;
        }

```

```

        ccode = NWSList(
            InxMSG("Choose A Country", 572),
            12, 40,
            rows,
            18,
            M_ESCAPE | M_SELECT,
            &listPtr,
            NUTHandle, NULL,
            NULL, NULL);

```

```

        /* Height */
        /* Width */

```

```

        if (ccode == M_SELECT)
        {

```

```

            if (NWSPushList(NUTHandle) != 0)
            {
                name = listPtr->text;

```

```

    ComputeGetRegion(name);
    NWSPopList(NUTHandle);
}
NWSDestroyList(NUTHandle);
closedir(dirDB);
goto getCountriesLoop;
)

NWSDestroyList(NUTHandle);
closedir(dirDB);
return;

errorNoDir:
NWSDestroyList(NUTHandle);
NWSAlert(12, 40, NUTHandle, InxMSG("Unable to locate Country directories
in SYS:DIRECPC\DB", 573));
return;
)

void DPCPointing(void)
(
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    NWInitList(NUTHandle, NULL);

    NWSAppendToList(MSG("Antenna Pointing Calculations", 574), (void *)1, NU
THandle);
    NWSAppendToList(MSG("Signal Strength Meter", 575), (void *)2, NUTHandle);
    while (ccode != M_ESCAPE)
    (
        ccode = NWSList(
            InxMSG("Dish Pointing", 576),
            12, 40,
            2,
            30,
            M_ESCAPE | M_SELECT,
            &listPtr,
            NUTHandle, NULL,
            NULL, NULL);

        /* Heigh
        /* Width

        if (ccode == M_SELECT)
        (
            if (NWSPushList(NUTHandle) != 0)
            (
                switch ((int)listPtr->otherInfo)
                (
                    case 1:
                        ComputeCoordinates();
                        break;

                    case 2:
                        SignalMeter();
                        break;

                    default:
                        break;
                )
            )
            NWSPopList(NUTHandle);
        )
    )
}
NWSDestroyList(NUTHandle);
}

void UpdateAdapterInformation(LONG portal)
(
    PCB
    int
    MUXdacau_t
    *dptr;
    BYTE
    char
    string(80);
    serialNumber(10);

    NWSGetPCB(&portalPtr, portal, NUTHandle);
    /* do generic stuff */
    line = 0;

    DIOGetSN(serialNumber);
    NWSShowPortalline
    (
        line++,
        GblDataCol,
        serialNumber,
        CStrLen(serialNumber),
        portalPtr
    );

    NWSShowPortalline
    (
        line++,
        GblDataCol,
        SiteID,
        CStrLen(SiteID),
        portalPtr
    );

    NWSprintf(string, MSG("%s", 503), CASDBdacau.entries == 0 ? MSG("FALSE",
474) : "TRUE");
    NWSShowPortalline
    (
        line++,
        GblDataCol,
        string,
        CStrLen(string),
        portalPtr
    );

    count = CASDBdacau.entries;
    NWSprintf(string, MSG("%d", 495), count);
    NWSShowPortalline
    (
        line++,
        GblDataCol,
        string,
        CStrLen(string),
        portalPtr
    );

    dptr = (MUXdacau_t *)CASDBdacau.p_buffer;
    while (count)
    (
        line++;
        if (line > (portalPtr->virtualHeight - 1))
            break;
    )
}

```

```

for (col = 8; col <= 64; col += 16, count--)
{
    if (!count)
        break;

    NWSprintf( string, MSG("%2.2X%2.2X%2.2X%2.2X", 504),
        dptr->groupid.i[2], dptr->groupid.i[1],
        dptr->groupid.i[0], dptr->version);

    NWSShowPortalLine(
        line,
        col,
        string,
        CStrLen(string),
        portalPtr
    );

    dptr++;
}

NWSUpdatePortal( portalPtr );

void DisplayAdapterInfo(void)
{
    int         line, len;
    BYTE        oldPortal;
    PCB         *portalPtr;
    BYTE        string[80];

    line = 5 + 3; /* Site ID, S/N, Key Status, Number of Communities,
        Current Communities */
    extra_community_lines = 0;
    line += (CASDBdcau.entries / 4);

    oldPortal = NUTHandle->currentPortal;
    NWSSelectPortal( NUTHandle );
    BackgroundPortal = NWSCreatePortal
    (
        1 /* gblUFFTopLine */,
        1 /* gblUFFLeftCol */,
        ((line + 4) > (ScreenHeight - 3)) ? ScreenHeight - 3 : line + 4,
        /* gblUFFHeight + gblUFFHeight */
        ScreenWidth - 2 /* gblUFFWidth */,
        line,
        ( ScreenWidth - 4 /* gblUFFWidth - 2 */ ),
        TRUE,
        MSG("DPC Adapter Information", 502),
        VNORMAL,
        SINGLE,
        VINTENSE,
        CURSOR_OFF,
        VIRTUAL,
        NUTHandle
    );
    if ( BackgroundPortal > MAXPORTALS )
    {
        NWSSelectPortal( oldPortal, NUTHandle );
        return;
    }

    NWSSetPCB( &portalPtr, BackgroundPortal, NUTHandle );
    portalPtr->showScrollBars = SHOW_VERTICAL_SCROLL_BAR;
    portalPtr->showScrollBars |= TEXT_SENSITIVE_SCROLL_BARS;
    portalPtr->verticalScroll = SCROLL_ON;

```

```

NWSClearPortal( portalPtr );

```

```

/*
 * Start filling in the static portal lines.
 */

```

```

line = 0;
NWSprintf(string, MSG("Serial Number
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Site ID
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Have Keys Status
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Number Of Communities
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

NWSprintf(string, MSG("Communities
len = CStrLen( string );
NWSShowPortalLine( line++, BORDER_WIDTH, string, len, portalPtr );

GblDataCol = BORDER_WIDTH + 24;

UpdateAdapterInformation(BackgroundPortal);
BackgroundFuncPtr = UpdateAdapterInformation;
HandleScrollablePortal(portalPtr);
BackgroundFuncPtr = NULL;
NWSDestroyPortal( BackgroundPortal, NUTHandle );
NWSSelectPortal( oldPortal, NUTHandle );

```

```

void DisplayMLIDState(void)
{

```

```

    int         line;
    int         count;
    int         numberOfGenerics;
    int         len;
    int         promptMax;
    struct DriverStatsStructure *stats;
    struct DriverConfigurationStructure *config;
    ProtocolNodeStructure *protocol;
    CustVars    *customPtr;
    BYTE        *customStrings, *ptr;
    BYTE        oldPortal;
    BYTE        name[128], *namePtr;
    PCB         *portalPtr;
    BYTE        string[80];

```

```

GblDataCol = ( ScreenWidth - 4 ) - BORDER_WIDTH - STATS_DATA_WIDTH;
if (DPCGetMLIDStats(&stats))
{
    return;
}

```

```

DIOGetMLIDConfig(&config);

```

```

/*
 * Build up the LAN name followed by its I/O resources

```

```

* to be used as the portals header.
*/

```

```

if (config->DLogicalName[0] != 0)
    NWSprintf(name, MSG("s %s", 270), config->DLogicalName, config
->DShortName);
else
    NWSprintf(name, MSG("s", 271), config->DShortName);

if (config->DIOPortsAndRanges[1] > 0)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" port=%X", 272), config->DIOPortsAndRang
es[0]);
}
if (config->DIOPortsAndRanges[3] > 0)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" port=%X", 273), config->DIOPortsAndRang
es[2]);
}
if (config->DMemoryDecodeAndLength[0].LANMemoryAddress > 0)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" memory=%X", 274),
        config->DMemoryDecodeAndLength[0].LANMemoryAddre
ss);
}
if (config->DMemoryDecodeAndLength[1].LANMemoryAddress > 0)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" memory=%X", 275),
        config->DMemoryDecodeAndLength[1].LANMemoryAddre
ss);
}
if (config->DIntLine[0] != 0xff)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" int=%X", 276), config->DIntLine[0]);
}
if (config->DIntLine[1] != 0xff)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" int=%X", 277), config->DIntLine[1]);
}
if (config->DDMAline[0] != 0xff)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" dma=%X", 278), config->DDMAline[0]);
}
if (config->DDMAline[1] != 0xff)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" dma=%X", 279), config->DDMAline[1]);
}
if (config->DMediaType != NULL)
{
    namePtr = name + CStrLen(name);
    NWSprintf(namePtr, MSG(" frame=%s", 280), config->DMediaType);
}
namePtr = name + CStrLen(name);
NWSprintf(namePtr, MSG(" ", 281));

```

```

/*
* Before we can create the portal, we must add up the number number
* of lines we'll need, which will be bigger than the window.
*/

```

```

line = 3;
/* lines for version, node address, and
protocol header */

/* Add up the number of protocols bound to this board */
protocol = MLIDProtocolListByBoard( DIOBoard );
while (protocol != NULL)
{
    ++line;
    protocol = (ProtocolNodeStructure *)protocol->ProtocolBoardLink;
}

/* Add in the number of generic statistics */
line += 2; /* Blank line and Generic statistics header */

numberOfGenerics = stats->GenericVariableCount;
line += numberOfGenerics;

promptMax = 0;
for (count = 0; count < numberOfGenerics ; count++)
{
    len = CStrLen( GetMsg(GenericDescriptionTable[count]) );
    if (promptMax < len)
    {
        promptMax = len;
    }
}

/* Add in the number of custom statistics */
line += 2; /* Blank line and Custom statistics header */

customPtr = (CustVars *)(&stats->CustomVariableCount);
customStrings = (BYTE *) (customPtr->CustomVariable[customPtr->CustomVari
ableCount]);
customStrings += sizeof(WORD);

line += customPtr->CustomVariableCount;

/* Check lengths of custom strings */

ptr = customStrings; /* temp pointer to walk down custom prompts */
for ( count = 0; count < customPtr->CustomVariableCount; count++)
{
    len = CStrLen( ptr );
    if ( promptMax < len )
        promptMax = len;
    while ( *( ptr++ ) )
        ; /* find the next string */
}

line += 1; /* add a blank line for the bottom */

if ( promptMax < 46 )
    promptMax = 64; /* minimum portal width */
else if ( promptMax > 60 )
    promptMax = 76; /* maximum portal width */
else
    promptMax += 16; /* custom portal width within range */

/* build the portal */

```



```

BORDER_WIDTH + INDENT_WIDTH,
customStrings,
CStrLen( customStrings ),
portalPtr

```

```

);
while ( *( customStrings++ ) )
/* find the next string */

```

```

UpdateStatsInformation( BackgroundPortal );
BackgroundFuncPtr = UpdateStatsInformation;
HandleScrollablePortal( portalPtr );
BackgroundFuncPtr = NULL;
NWSDestroyPortal( BackgroundPortal, NUTHandle );
NWSSelectPortal( oldPortal, NUTHandle );

```

```

LONG DisconnectRoutine( FIELD *fp, int key, int *changed, NUTInfo *handle )
{

```

```

fp = fp;
key = key;
changed = changed;
handle = handle;

```

```

DloEndConn();

```

```

return( K_NORMAL );

```

```

LONG DialInternetRoutine( FIELD *fp, int key, int *changed, NUTInfo *handle )
{

```

```

fp = fp;
key = key;
changed = changed;
handle = handle;

```

```

DloStartConn( DLO_INET_TIMEOUT );

```

```

return( K_NORMAL );

```

```

LONG DialPDRoutine( FIELD *fp, int key, int *changed, NUTInfo *handle )
{

```

```

fp = fp;
key = key;
changed = changed;
handle = handle;

```

```

DloStartConn( DLO_PACKAGE_TIMEOUT );

```

```

return( K_NORMAL );

```

```

LONG SendModemRoutine( FIELD *fp, int key, int *changed, NUTInfo *handle )
{

```

```

BYTE sendStr[82];
int ccode;
LONG len;
fp = fp;
key = key;

```

```

changed = changed;
handle = handle;

```

```

sendStr[0] = 0;

```

```

if ( !NWSPushList( NUTHandle ) )
return( K_NORMAL );

```

```

ccode = NWSEditString(
12, 40,
/* center line, column */
1, 40,
/* edit height, width */

```

```

InxMSG( "Modem Send Editor", 587 ), /* header

```

```

InxMSG( "Send : ", 588 ), /* prompt

```

```

(BYTE*) &sendStr, 80, /*
EF_ANY, NUTHandle, /* type handle

```

```

NULL, NULL,

```

```

/* insert Proc, action Proc */
MSG( "a.z0..9A..Z.-", 589 );
if ( ccode & E_ESCAPE )
{

```

```

/* Start Change by DMH 961115 */
NWSPopList( NUTHandle );
/* End change by DMH 961115 */
return( K_NORMAL );
}

```

```

if ( ( len = CStrLen( sendStr ) ) )
{

```

```

DloSend( sendStr, len, DLO_INET_TIMEOUT );

```

```

DloSend( MSG( "\r", 609 ), 1, DLO_INET_TIMEOUT );

```

```

NWSPopList( NUTHandle );
return( K_NORMAL );
}

```

```

void ModemControl( void )
{

```

```

int i;

```

```

NWSInitForm( NUTHandle );

```

```

i = 0;

```

```

NWSAppendHotSpotField( i, 15-(20/2), NORMAL_FIELD, MSG( "Disconnect the Mo
dem", 547 ),

```

```

i++;
DisconnectRoutine, NUTHandle );

```

```

NWSAppendHotSpotField( i, 15-(18/2), NORMAL_FIELD, MSG( "Dial the Internet
", 549 ),

```

```

DialInternetRoutine, NUTHandle );

```

```

i++;
NWSAppendHotSpotField( i, 15-(26/2), NORMAL_FIELD, MSG( "Dial the Package
Delivery", 550 ),

```

```

DialPDRoutine, NUTHandle );

```

```

i++;
NWSAppendHotSpotField( i, 15-(18/2), NORMAL_FIELD, MSG( "Send to the Modem

```

```

", 551),

    SendModemRoutine, NUTHandle);

    i++;

    NMSeditPortalForm(InxMSG("Modem Control Options", 552),
        10, 30,
        i, 30,
        /* form height, width */
        F_NOVERIFY, F_NO_HELP,
        /* Control flags, help m
        message */
        InxMSG("Save Changes?", 585),
        NUTHandle);

    /* Confirm message, hand
    le */

    NMSDestroyForm(NUTHandle);

    /*.....*/
    MainOptionsHandler(void)

    Description:
    * This routine is where the main agent thread lives.
    * It initiates the NUT screen, waits for the DPC MLID
    * to become active, and wait for user commands.

    Input: nothing

    Output: nothing

    Returns: nothing

    /*.....*/
    void MainOptionsHandler()
    (
        int choice, prevChoice, i;
        int exitFlag = FALSE;
        LIST *defaultList;
        LONG type;
        BYTE value;
        int countdown = MAX_COUNTDOWN;
        LONG mainPortal;
        LONG removedCount;

        /*if DRIVER_IO
        #else
        void (*ControlEntryPoint) () = NULL;
        LONG board;
        struct DriverConfigurationStructure *config;
        char *configName;

        #endif

        /*
        * Clear the screen by creating huge portal with VNORMAL attribute.
        */

        GetScreenSize(&ScreenHeight, &ScreenWidth);
        ScreenHeight = ScreenHeight - NUTHandle->headerHeight;
        mainPortal = NMScreatePortal(
            NUTHandle->headerHeight, 0,

            /* line, column

```

```

        width
        /* frame height/
        */
        ScreenHeight, ScreenWidth,

        /* virtual height
        */
        ScreenHeight, ScreenWidth,

        /* virtual height
        */
        SAVE, NULL,
        VNORMAL, NOBORDER,
        /* Save flag, header text
        */
        r attr, border type
        VNORMAL, CURSOR_OFF,
        /* head
        */
        r attr, cursor flag
        VIRTUAL, NUTHandle);
        /* borde
        */
        t flag, handle
        /* direc
        */

        if (mainPortal >= MAXPORTALS)
            return;

        #if DRIVER_IO
        LookForAdapter:
        #endif

        while (NMSKeyStatus(NUTHandle))
            NMSGetKey(&type, &value, NUTHandle);

        NMSUpdatePortal( NUTHandle->portal[mainPortal] );

        /*
        * Display our server name in an info portal at the top of the screen.
        */

        UpdateHelpPortal();

        /*
        * Lets look for a DPC adapter.
        */

        StartWaitWithMessage(ScreenHeight/2, ScreenWidth/2, NUTHandle, InxMSG("
        Waiting for DPC adapter to load", 143));
        DPCName[0] = 3;

        #if DRIVER_IO

        while(DIORRegisterWithAdapter(DPCName) && exitFlag == FALSE)
        (
            delay(500);
            if (NMSKeyStatus(NUTHandle))
            (
                NMSGetKey(&type, &value, NUTHandle);
                if ((type == K_ESCAPE) || (type == K_AF10))
                (
                    NMSendWait(NUTHandle);
                    return;
                )
            )
            if (--countdown == 0)
            (
                Spin(NUTHandle);
                countdown = MAX_COUNTDOWN;
            )
        )
        if (exitFlag)
            return;
        removedCount = DIORemovedCount;
        choice = prevChoice = PackageDelivery ? 1 : 2;

        #else
        while(1)
        (
            for(board = 0; board < NumberOfLANs; board++)

```

```

oint);
    CLSGetMLIDControlEntry(board, (void(*)())&ControlEntryP
    if (ControlEntryPoint)
    {
        config = (struct DriverConfigurationStructure *)
            CommandMlid(board
    d, 0, (LONG)ControlEntryPoint);
        if (config)
        {
            configName = config->DShortName;
            if (!CStrCmp(configName, DPCName))
                goto FoundDPCBoardNumber;
        }
    }
    delay(500);
    if (NWSKeyStatus(NUTHandle))
    {
        NWSGetKey(&type, &value, NUTHandle);
        if ((type == K_ESCAPE) || (type == K_AFI0))
        {
            NWSEndWait(NUTHandle);
            return;
        }
        if (--countdown == 0)
        {
            Spin(NUTHandle);
            countdown = MAX_COUNTDOWN;
        }
    }
}
#endif
/*
 * OK. We have a DPC MLID. Lets set up the main menu and
 * wait for the user to do something.
 */
#endif
#endif
FoundDPCBoardNumber:
    NWSEndWait(NUTHandle);
    NWSEnableInterruptKey(K_AFI0, ExitHandler, NUTHandle);
    /*
     * Initialize the main options menu.
     */
    NWSInitDhList(NUTHandle, Free); /* Don't sort menu items. */
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_ONE,
        (BYTE *) "Package Delivery", &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_TWO,
        MSG("Display MLID Stats", 102), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_THREE,
        MSG("DPC Configuration", 95), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_FOUR,
        MSG("Dish Pointing", 344), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_FIVE,
        MSG("Adapter Information", 150), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_SIX,
        MSG("Modem Control", 501), &NUTHandle->messages);
    NWSSetDynamicMessage(DYNAMIC_MESSAGE_SEVEN,

```

```

MSG("-Exit DPCAGENT", 586), &NUTHandle->messages);
    if (PackageDelivery)
        NWSAppendToMenu(DYNAMIC_MESSAGE_ONE, 1, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_TWO, 2, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_THREE, 3, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_FOUR, 4, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_FIVE, 5, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_SIX, 6, NUTHandle);
    NWSAppendToMenu(DYNAMIC_MESSAGE_SEVEN, 7, NUTHandle);
    while (exitFlag == FALSE)
    {
        prevChoice = choice;
        /* Set defaultList to previous choice */
        defaultList = NWSGetListHead(NUTHandle);
        if (!PackageDelivery)
            choice = 1;
        for (i = choice - 1; i; i--)
            defaultList = defaultList->next;
        choice = NWSMenu(InxMSG("DPCAGENT Options", 151),
            10, 40, defaultList, NULL, NUTHandle, NULL);
        if (removedCount != DIORemovedCount)
        {
            NWSDestroyMenu(NUTHandle);
            NWSClearPortal(NUTHandle->portal[mainPortal]);
            goto LookForAdapter;
        }
        switch (choice)
        {
            case 1:
                if (NWSPushList(NUTHandle))
                {
                    DisplayPDInterface();
                    NWSPopList(NUTHandle);
                }
                break;
            case 2:
                /* Display MLID Stats */
                if (NWSPushList(NUTHandle))
                {
                    DisplayMLIDStats();
                    NWSPopList(NUTHandle);
                }
                break;
            case 3:
                /* Modem Configuration */
                if (NWSPushList(NUTHandle))
                {
                    DPCConfiguration();
                    NWSPopList(NUTHandle);
                }
                break;
            case 4:
                /* Signal Strength Meter */
                if (NWSPushList(NUTHandle))
                {
                    DPCPointing();
                    NWSPopList(NUTHandle);
                }
                break;
            case 5:
                /* Display Adapter Information */
                if (NWSPushList(NUTHandle))
                {
                    DisplayAdapterInfo();
                    NWSPopList(NUTHandle);
                }

```

```

    break;

case 6:
    /* Display Adapter Information */
    if (NWSPushList(NUTHandle)) {
        ModemControl();
        NWSPopList(NUTHandle);
    }
    break;

default:
    if (NWSConfirm(InxMSG("Exit DPCAGENT?", 152), 0, 0, TRUE, NULL,
        NUTHandle, NULL) == TRUE)
        exitFlag = TRUE;
    else
        if (choice != 7)
            choice = prevChoice;
    }

NWSDestroyMenu(NUTHandle);
}

/*.....*/
DPCAgentMain(void *parm)
{
    Description:      Main thread. It will initialize the NUT screen and wait
                      user input.

    Input:            parm
                      - ignored

    Output:           Nothing

    Returns:          Nothing
}

void DPCAgentMain(void *parm)
{
    parm = parm;

    MainOptionsHandler();

    ReturnResources(1);
    exit(1);
}

/*.....*/
main(int argc, char *argv[])
{
    Description:      Initialization routine.

    Input:            Nothing

    /* Create a screen for displaying our information */
}

/*.....*/
LONG ScreenID;
LONG main(int argc, char *argv[])
{
    LONG currentScreen;
    LONG ser[2];
    LONG ccode;
    int i;

    for (i = 1; i < argc; i++)
    {
        if (ICmpB(argv[i], MSG("-DEBUG", 182), 6) == -1)
        {
            if ((DebugFlag = strtol(argv[i][6], 0, 16)) == 0)
                DebugFlag = TRUE;
        }
    }

    /* Get a handle for allocating a resource tag */
    NLMHandle = (struct LoadDefinitionStructure *)GetNLMHandle();
    if (!NLMHandle)
        return(-1);

    if (!ReturnMessageInformation((LONG)NLMHandle, (BYTE ***)&NLMMessageTabl
e, NULL, NULL, NULL))
        return(-1);

    OSGetCountryInfo(&GblDOSCountryInfo);

    /* Allocate a resource tag to use for memory allocations */
    allocrTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Memory", 167
),
    ),
    AESTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT AES", 1
), AESProcessSignature);
    asyncIoTTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Async I/O",
169),
    ASYNCIOSignature);
    timerTag = AllocateResourceTag(NLMHandle, TxtMSG("DPCAGENT Delay Timer",
170),
    TimerSignature);

    if (allocrTag == NULL ||
        AESTag == NULL ||
        timerTag == NULL ||
        ASYNCIOSignature == NULL)
    {
        ConsolePrintf(TxtMSG("DPCAGENT: Unable to allocate Resource Tags
", 171));
        return(-1);
    }
}

```

```

currentScreen = GetCurrentScreen();
SetAutoScreenDestructionMode(TRUE);

/* Initialize the screen interface */
ScreenID = CreateScreen("DPCAgent Utility", AUTO_DESTROY_SCREEN);
ccode = NWSInitializeNut(InxMSG("DPC AGENT PROGRAM", 172), AGENT_VERSION);

if (ccode)
{
    SMALL_HEADER, NUT_REVISION_LEVEL, NULL, NULL,
    ScreenID, (LONG)allocrTag, &NUTHandle);

    ConsolePrintf(TxtMSG("DPCAGENT: Unable to initialize NUT.", 174))
    return(-1);
}

// NLMMessageTable = (BYTE **)&(NUTHandle->messages);

/* Get a connection with the server we're on */
if (DebugFlag) {
    DisplayScreen(ScreenID);
    SetCurrentScreen(currentScreen);
}

#ifdef LOG_ECB_ACTIVITY
    if (DebugFlag >= 0x51) {
        DPC_TCID = GetThreadGroupID();
        LogRegisterClient("SYS:DIRECTPC/LOG.CFG", 2048, &LogClientHandle);
        LogRegisterEvent("ECB", &LogECBHandle);
    }
#endif /* LOG_ECB_ACTIVITY */

else {
    DestroyScreen(currentScreen);
}

#ifdef DEBUG_ALL
    if (OpenScreen(MSG("DPCAGENT Debug Screen", 224), screenTag, &DebugScreenID))
    {
        ConsolePrintf(MSG("DPCAGENT: Unable to create debug screen.", 225));
        return(-1);
    }
#endif

DPCUpdateConfig();

InetChangeProtocol();

DPCSetMaxConnections(ser);

DPCAgentPID = BeginThread(DPCAgentMain, NULL, NULL, NULL);
RenameThread(DPCAgentPID, "DPCAgent Main");
if (PackageDelivery) {
    DPCFilePID = BeginThread(DPCFileMain, NULL, 32 * 1024, NULL);
    RenameThread(DPCFilePID, "DPCAgent PD");
    DPCAccessPID = BeginThread(DPCAccessMain, NULL, NULL, NULL);
    RenameThread(DPCAccessPID, "DPCAgent Access");
}

DPCModemPID = BeginThread(DPCModemMain, NULL, NULL, NULL);
RenameThread(DPCModemPID, "DPCAgent Modem");
if (DPCMaxConnections) {
    DPCInetPID = BeginThread(DPCInetMain, NULL, NULL, NULL);
    RenameThread(DPCInetPID, "DPCAgent Tinet");
}

signal(SIGTERM, ReturnResources);
ExitThread(TSR_THREAD, 0);

```

```

ReturnResources(int sig)
Description:
Shutdown routine. Returns the modules resources.

Input:
sig
Output:
Nothing
Returns:
Nothing

```

```
void ReturnResources(int sig)
{
    int i;
    int countdown = MAX_COUNTDOWN;

    sig = sig;
    ExitingFlag = TRUE;
    if (InReturnResources)
        return;

    InReturnResources = TRUE;

```

```

/* Force NUT to escape out of all menus so that it can clean
 * before we call NWSRestoreNUT(NUT has a bug were it will
 * attempt to free up its memory twice(ABEND) if the user
 * leaves the screen a couple of menus in before unloading
 * the application from the command line.
 */
for(i = 0; i < 4; i++)
    NWSUngetKey(UGK_ESCAPE_KEY, UGK_NORMAL_KEY, NUTHandle);

StartWaitWithMessage(ScreenHeight/2, ScreenWidth/2, NUTHandle,
    InxMSG("Waiting for threads to Exit", 226));

if (AccessAsleep)
    ResumeThread(DPCAccessPID);

if (InetAsleep)
    ResumeThread(DPCInetPID);

```

```

/* Give threads and NUT a chance to execute.
 * Wait 1.5 seconds since signal meter and stats could be sleeping
 * for up to a second.
 */
delay(1500);

```

```

while(DPCFilePID || DPCModemPID || DPCAccessPID || DPCInetPID) {
    void DPCPDterminate(void);
    DPCPDterminate();

    if (AccessAsleep)
        ResumeThread(DPCAccessPID);
}

```

Thu Jul 17 14:46:11 1997

dpcagent.c

Page 51

```
if (InetAsleep)
    ResumeThread(DPCInetPID);

if (--countdown == 0) (
    Spin(NUTHandle);
    countdown = MAX_COUNTDOWN;
)
    ThreadSwitchWithDelay();
)

#ifdef LOG_ECB_ACTIVITY
    if (DebugFlag >= 0x51) (
        LogDeregisterEvent(&LogECBHandle);
        LogDeregisterClient(&LogClientHandle);
    )
    #endif /* LOG_ECB_ACTIVITY */

    if (DlCfg.out_protocol == OUT_PPP)
    (
        DisconnectPPP();
    )
    DIODERegisterAgent();

#ifdef DEBUG_ALL
    CloseScreen(DebugScreenID);
#endif
    NWSendWait(NUTHandle);
    NWSRestoreNut(NUTHandle);
)
)
```

```

/*include "dpcagent.h"      /* Our header file */

LONG
int
/*
 * Access Configuration Variables
 */
BYTE SiteID[9];
WORD CDBVersion = 0;
WORD CDBETHVersion = 0;
BYTE DacauFlag = 0;
LONG DacauTime = 0;
LONG RndDacauTime = 0;
DCAUrequest_t DacauRequest;
CASDBbuffer CASDBpacau;
CASDBbuffer CASDBpbe;
CASDBbuffer CASDBdacau;
CASDBbuffer CASDBbecau;

/* Elements tables */
static CDBelement_t Elements[MAXELEMENTS];
static CDBelement_t UpdatedElements[MAXELEMENTS];

BYTE *RcvBuf;
MACrecord_t CASmacs[MAX_MAC_RECORDS];
BYTE AccessAddress[] = {0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
/* Access thread needs t
o wake up flag */

BYTE
/* Stores current packet
 */
/* ECB linked list variables.
 */
static ECB *AccessECBHead = 0; /* Take ECBs from here */
static ECB *AccessECBTail = 0; /* Put ECBs here */

/*
 * element key used by IroSetAddress.
 */
BYTE MagicKey[] = { 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11, 0x11 };

/*
 * Address Type Table used by MACbuildAddr().
 */
static unsigned char table[5][2] = {
    {
        /* Hybrid Internet */
        /* CAS individual */
        /* Package Delivery */
        /* Data Feed */
        /* BYPASS */
    },
    {
        /* MAC_INDIVID, MAC_NORMAL */
        /* MAC_INDIVID, MAC_BYPASS */
        /* MAC_MULTICAST, MAC_NORMAL */
        /* MAC_MULTICAST, MAC_NORMAL */
        /* MAC_MULTICAST, MAC_BYPASS */
    },
    {
        /* MAC_INDIVID, MAC_NORMAL */
        /* MAC_INDIVID, MAC_BYPASS */
        /* MAC_MULTICAST, MAC_NORMAL */
        /* MAC_MULTICAST, MAC_NORMAL */
        /* MAC_MULTICAST, MAC_BYPASS */
    },
    {
        /* MAC_INDIVID, MAC_NORMAL */
        /* MAC_INDIVID, MAC_BYPASS */
        /* MAC_MULTICAST, MAC_NORMAL */
        /* MAC_MULTICAST, MAC_NORMAL */
        /* MAC_MULTICAST, MAC_BYPASS */
    },
    {
        /* MAC_INDIVID, MAC_NORMAL */
        /* MAC_INDIVID, MAC_BYPASS */
        /* MAC_MULTICAST, MAC_NORMAL */
        /* MAC_MULTICAST, MAC_NORMAL */
        /* MAC_MULTICAST, MAC_BYPASS */
    }
};

BYTE SerialNum[9];
BYTE SerialNumPacked[3];

/*
 * ReadConfig(void)
 */
Description: This routine reads the DPC.CFG file and stores the conte
nts

```

into the appropriate globale variables.

Input: Nothing

Output: Nothing

Returns: Nothing

static void ReadConfig(void)

```

{
    int handle, k;
    BYTE *mem_ptr;

    for(k = 0; k < MAXELEMENTS; k++)
    {
        Elements[k].in_use = 'N';
        UpdatedElements[k].in_use = 'N';
    }
}

```

handle = open(MSG("SYS:DIREPCP\\DB\\DPC.CFG", 203), O_RDONLY);

if (handle != -1)

```

{
    read(handle, &SiteID, sizeof(SiteID));
    read(handle, &CDBVersion, sizeof(CDBVersion));
    read(handle, &CDBETHVersion, sizeof(CDBETHVersion));
    read(handle, &DacauFlag, sizeof(DacauFlag));
    read(handle, &DacauTime, sizeof(DacauTime));
    read(handle, &DacauRequest, sizeof(DCAUrequest_t));
    read(handle, &CASDBpacau, sizeof(CASDBbuffer));
    read(handle, &CASDBpbe, sizeof(CASDBbuffer));
    read(handle, &CASDBdacau, sizeof(CASDBbuffer));
    read(handle, &CASDBbecau, sizeof(CASDBbuffer));
}
else

```

UpdateFileStatus(MSG("Obtaining Encryption Keys", 204));

```

for(k = 0; k < MAX_MAC_RECORDS; k++)
    CASmacs[k].in_use = 'N';

```

if(handle != -1)

```

{
    close(handle);
}
else

```

```

{
    SiteID[0] = '\0';
    CASDBpacau.version = CASDBpbe.version = 0;
    CASDBdacau.version = CASDBbecau.version = 0;
    CASDBpacau.entries = CASDBpbe.entries = 0;
    CASDBdacau.entries = CASDBbecau.entries = 0;
}

```

```

CASDBpacau.entry_len = PACAU_LEN;
CASDBpbe.entry_len = PEB_LEN;
CASDBdacau.entry_len = DACAU_LEN;
CASDBbecau.entry_len = ECAU_LEN;

```

if (DacauFlag)

```

{
    UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 257));
    WaitingForKeys = TRUE;
}

```

```
DacauRequest.opcode = DACAU_REQUEST;
RndDacauTime = time(0) + rand();
```

```
.....
```

```
SaveConfig(void *parm)
```

```
Description:
```

```
anytime      This routine writes the access structures out to DPC.CFG
              a change is made to any of the structures.
```

```
Input:      Nothing
```

```
Output:     Nothing
```

```
Returns:    Nothing
```

```
.....
```

```
static void SaveConfig(void)
```

```
{
```

```
    int handle;
```

```
    int tmpFlag;
```

```
    handle = open(MSG("SYS:DIRECPC\\DB\\DPC.CFG", 206), O_RDWR | O_CREAT, S_
```

```
IWRITE |
```

```
S_IREAD);
```

```
    tmpFlag = DacauFlag;
```

```
    if (WaitingForKeys)
```

```
    {
```

```
        DacauFlag = 1;
```

```
    }
```

```
    /* Write version */
```

```
    write(handle, SiteID, sizeof(SiteID));
```

```
    write(handle, &CDBVersion, sizeof(CDBVersion));
```

```
    write(handle, &CDBEthVersion, sizeof(CDBEthVersion));
```

```
    write(handle, &DacauFlag, sizeof(DacauFlag));
```

```
    write(handle, &DacauTime, sizeof(DacauTime));
```

```
    write(handle, &DacauRequest, sizeof(DACAUrequest_t));
```

```
    /* Write Info headers */
```

```
    write(handle, &CASDBpacau, sizeof(CASDBbuffer));
```

```
    write(handle, &CASDBpeb, sizeof(CASDBbuffer));
```

```
    write(handle, &CASDBdacau, sizeof(CASDBbuffer));
```

```
    write(handle, &CASDBbecau, sizeof(CASDBbuffer));
```

```
    /* Write buffers */
```

```
    close(handle);
```

```
    DacauFlag = tmpFlag;
```

```
.....
```

```
AccessESR(ECB *ecb)
```

```
Description:
```

```
This routine is called by the MLID interrupt service
routine when a packet is received. The ECB is queued
and if the Access File thread is sleeping, it is
```

```
.....
```

```
woken up.
```

```
Input:      ecb
```

```
the packet
```

```
Output:     nothing
```

```
Returns:    0
```

```
.....
```

```
int      AccessESR(ECB *ecb)
{
    /* Link the ECB to the Tail of the linked list */
    ecb->ECB_NextLink = 0;
    if (AccessECBTail)
        AccessECBTail->ECB_NextLink = ecb;
    AccessECBTail = ecb;
    if (AccessECBHead == 0)
        AccessECBHead = ecb;

    /* Wake up file thread only if we need to */
    if (AccessAasleep)
        Resumethread(DPCAccessPID);

    #ifdef LOG_ECB_ACTIVITY
        if (LogECBHandle) {
            int TGID = SetThreadGroupID(DPC_TGID);
            LogMsg(LogClientHandle, LogECBHandle, FALSE,
                "ACCESS queue(%08lx)\n", ecb);
            SetThreadGroupID(TGID);
        }
    #endif /* LOG_ECB_ACTIVITY */

    return(0);
}
.....
find_peb(ID element_pattern,
          char ver_pattern)
.....
Description:      This routine searches the PEB list to find an entry that
                  matches the element and version pattern passed in.
Input:            element_pattern
                  ver_pattern
to search for
Output:           nothing
Returns:          pointer to peb entry if it was found
                  otherwise its a NULL
.....
MUXecau_t *find_ecau(ID group_pattern, char ver_pattern)
{
    .....
}
```

```
.....
```

```

int i;
MUXecau_t *p_ecau, *ret = NULL;

for(i = 0; i < CASDBecau.length; i += ECAU_LEN)
{
    p_ecau = (MUXecau_t *) (CASDBecau.p_buffer + i);
    if(CCmpB(&p_ecau->groupid, &group_pattern, sizeof(ID)) == -1)
    {
        if(ver_pattern == -1)
        {
            ret = p_ecau;
            break;
        }
        else
        {
            if(ver_pattern == p_ecau->version)
            {
                ret = p_ecau;
                break;
            }
        }
    }
    return(ret);
}

static MUXpeb_t *find_peb(ID element_pattern, char ver_pattern)
{
    unsigned short i, num_addr;
    MUXpeb_t *p_peb, *ret = NULL;

    for(i = 0; i < CASDBpeb.length; i += PEB_LEN)
    {
        p_peb = (MUXpeb_t *) (CASDBpeb.p_buffer + i);
        if(CCmpB(&p_peb->elementid, &element_pattern, sizeof(ID)) == -1)
        {
            if(ver_pattern == -1)
            {
                ret = p_peb;
                break;
            }
            else
            {
                if(ver_pattern == p_peb->version)
                {
                    ret = p_peb;
                    break;
                }
            }
        }
        num_addr = p_peb->num_addr(0);
        num_addr |= ((unsigned short)p_peb->num_addr(1)) << 8;
        i += num_addr * MAC_LENGTH;
    }
    return(ret);
}

/*
 * Deletes element not only from the CASDB
 * but from adapter as well
 */
.....
del_peb_element(int e_num)
.....

```

Description:

This routine deletes and from the CASDB and from the adapter.

Input:

e_num

delete

Output:

nothing

Returns:

0

- index of the element t

```

...../
static del_peb_element(int e_num)
{
    int k;
    DIDeleteAddress(Elements[e_num].channel, (BYTE *)&Elements[e_num].e_mac
    );
    for(k = 0; k < MAX_MAC_RECORDS; k++)
    {
        if(CASmacs[k].in_use == 'Y' &&
            CCmpB(&CASmacs[k].dpc_mac, &Elements[e_num].e_mac, sizeof(MACadd
            r_t)) == -1)
        {
            CASmacs[k].in_use = 'N';
            break;
        }
        Elements[e_num].in_use = 'N';
        return(0);
    }
    .....
    replace_peb_element(int num,
        unsigned char new_ver)
    .....
    Description: This routine replaces the version numbers of the element
        indexed by num.
    .....
    Input: int_num
        nt to modify new_ver
        stuff into Element entry
    .....
    Output: nothing
    .....
    Returns: 0
    .....
    static replace_peb_element(int num, unsigned char new_ver)
    {
        int k;

```

- Index of Eleme

- new version to

static replace_peb_element(int num, unsigned char new_ver)

int k;

```

for(k = 0; k < MAX_MAC_RECORDS; k++)
{
    if(CASmacs[k].in_use == 'Y' &&
        CCmpB(&CASmacs[k].dpc_mac, &Elements[num].e_mac, sizeof(MACAddr_
t)) == -1)
    {
        CASmacs[k].dpc_mac.Ver = new_ver;
        break;
    }
    Elements[num].e_mac.Ver = new_ver;
    Elements[num].e_ver = new_ver;
    return 0;
}

```

```

/.....

```

```

find_pacau(ID group_pattern,
char ver_pattern)

```

```

Description: This routine attempts to locate a pacau given a group
pattern.

```

```

Input: group_pattern
ver_pattern
pattern
- group pattern to match
- version portion of the

```

```

Output:
Returns: pointer to pacau if successful
Otherwise NULL

```

```

/.....

```

```

MUXpacau_t *find_pacau(ID group_pattern, char ver_pattern)

```

```

int i;
MUXpacau_t *p_pacau, *ret = NULL;
for(i = 0; i < CASDBpacau.length; i += PACAU_LEN)
{
    p_pacau = (MUXpacau_t *) (CASDBpacau.p_buffer + i);
    if((CCmpB(&p_pacau->groupid, &group_pattern, 3)) == -1)
    {
        if(ver_pattern == -1)
        {
            ret = p_pacau;
            break;
        }
        else
        {
            if(ver_pattern == p_pacau->version)
            {
                ret = p_pacau;
                break;
            }
        }
    }
    return(ret);
}

```

```

/.....

```

```

find_dacau(ID group_pattern,
char ver_pattern)

```

```

Description: This routine attempts to locate a dacau given a group
pattern.

```

```

Input: group_pattern
ver_pattern
pattern
- group pattern to match
- version portion of the

```

```

Output:

```

```

Returns: pointer to dacau if successful
Otherwise NULL

```

```

/.....

```

```

MUXdacau_t *find_dacau(ID group_pattern, char ver_pattern)

```

```

int i;
MUXdacau_t *p_dacau, *ret = NULL;
for(i = 0; i < CASDBdacau.length; i += DACAU_LEN)
{
    p_dacau = (MUXdacau_t *) (CASDBdacau.p_buffer + i);
    if((CCmpB(&p_dacau->groupid, &group_pattern, 3)) == -1)
    {
        if(ver_pattern == -1)
        {
            ret = p_dacau;
            break;
        }
        else
        {
            if(ver_pattern == p_dacau->version)
            {
                ret = p_dacau;
                break;
            }
        }
    }
    return(ret);
}

```

```

reverse_key(BYTE *key)

```

```

Description: This routine swaps the byte order of the 8 byte
key passed in.

```

```

Input: key

```

```

Output:

```

```

Returns: pointer to pacau if successful
Otherwise NULL

```

```

- key to reverse

```

```
void reverse_key(BYTE *key)
```

```
{
    unsigned char x;
```

```
    x = key[0]; key[0] = key[1]; key[1] = x;
    x = key[2]; key[2] = key[3]; key[3] = x;
    x = key[4]; key[4] = key[5]; key[5] = x;
    x = key[6]; key[6] = key[7]; key[7] = x;
}
```

```
make_element_id(BYTE *e_id,
                char *e_id_txt)
```

```
Description: This routine is called by LuoSetAddress to help
              build the element id.
```

```
Input: e_id
```

```
       element id
```

```
Output: e_id_txt
```

```
       element text
```

```
Returns: nothing
```

```
void make_element_id(BYTE *e_id, char *e_id_txt)
```

```
    BYTE work[3];
    static char HexChar[] = MSG("0123456789ABCDEF", 208);
    unsigned char Ch, *p;
    int count = 3, i = 0;
```

```
    work[0] = e_id[2];
    work[1] = e_id[1];
    work[2] = e_id[0];
```

```
    p = work;
    while (count--)
```

```
    {
        Ch = *p++;
        e_id_txt[i++] = HexChar[Ch>>4]; /* high nibble */
        e_id_txt[i++] = HexChar[Ch & 0x0f]; /* low nibble */
    }
    e_id_txt[i] = '\0';
}
```

```
int43(LONG id, BYTE *array)
```

```
Description: This routine is called by MACbuildAddr to convert
              an id to its 3 byte equivalent.
```

```
Input: id
```

```
       - id to convert
```

```
r to 3 byte array
```

```
Output: array is filled in
```

```
Returns: 0 if successful
```

```
static int43(LONG id, BYTE *array)
```

```
{
    union {
        BYTE b[4];
        LONG w;
    } offset;
    int status = 0;

    offset.w = id;
    if (offset.b[3]==0 && !(offset.b[2] & 0xc0)) {
        array[0] = offset.b[2];
        array[1] = offset.b[1];
        array[2] = offset.b[0];
    }
    else {
        array[0]=array[1]=array[2] = 0;
        status = -1;
    }
    return status;
}
```

```
MACbuildAddr(char *element_txt,
              int feature,
              BYTE ver,
              MACAddr_t *address)
```

```
Description: This routine is called by LuoSetAddress to build a
              MAC address out of a file_id and community id.
```

```
Input: element_txt
       feature
```

```
AC_PKG, MAC_DF
```

```
_BYPASS_MULTICAST
ver
```

```
community id
```

```
ing address
```

```
Output: address is filled in
```

```
Returns: 0 if successful
```

```
int MACbuildAddr(char *element_txt, int feature, BYTE ver, MACAddr_t *address)
{
    BYTE element[3];
    LONG i;
```

- file id array
- MAC_HI, MAC_CA, M

or MAC

- low byte of co

- where store the result

```

int k, status = 0;

if(feature < 0 || feature > 5)
    return(-1);
/* Step one */
sscanf(element_txt, MSG("%lx", 137), &i);
if((status = int43(i, element)) != 0)
    return(status);
/* Step two */
for(k=0; k<3; k++)
{
    element[k] = element[k] << 2;
    element[k] |= ((k == 2) ? 0x00 : element[k+1]) >> 6;
}
/* Step three */
address->Element[0] = element[2];
address->Element[1] = element[1];
address->Element[2] = element[0];
/* Set Multicast/Individual */
if(table[feature][0] == MAC_MULTICAST)
    address->Element[0] |= MAC_MULTICAST;
/* Set Bypass/Normal */
if(table[feature][1] == MAC_BYPASS)
    address->Element[0] |= MAC_BYPASS;
/* Set application ID or Version number */
switch(feature)
{
    case MAC_HI:
        address->Ver = 0x02;
        break;
    case MAC_CAS_IND:
        address->Ver = 0x01;
        break;
    case MAC_BYPASS_MULTICAST:
        address->Ver = 0x00;
        break;
    default:
        address->Ver = ver;
        break;
}
/* Set reserved field */
address->Reserved[0] = address->Reserved[1] = 0x00;
if(feature == MAC_DF)
    address->Element[2] = 0xff;
return(status);
}

.....

static find_peb_mac(MACAddr_t mac_pattern)
{
    Description:    Finds the PEB entry which contains the mac address passed in.
    Input:          mac_pattern
                   - MAC address to search for
    Output:         Nothing
    Returns:        NULL if no entry found
                   Otherwise its a pointer to the PEB entry
}

```

```

static MUXpeb_t *find_peb_mac(MACAddr_t mac_pattern)
{
    unsigned short i, j, num_addr;
    MUXpeb_t *p_peb, *ret = NULL;
    for(i = 0; i < CASDBpeb.length; i += PEB_LEN)
    {
        p_peb = (MUXpeb_t *) (CASDBpeb.p_buffer + i);
        num_addr = p_peb->num_addr(0);
        num_addr |= ((unsigned short)p_peb->num_addr(1)) << 8;
        for(j = 0; j < num_addr; j++)
        {
            if(CCompB(&mac_pattern,
                _LENGTH, MAC_LENGTH) == -1)
            {
                ret = p_peb;
                goto peb_mac_exit;
            }
            i += num_addr * MAC_LENGTH;
        }
        peb_mac_exit:
        return(ret);
    }
    /*.....
    *
    * find_element_id(ID id,
    *                 BYTE ver)
    *
    * Description:    Find the elements index into the Element table.
    *
    * Input:          id
    *                 ver
    *
    * Output:         Nothing
    *
    * Returns:        -1 if not found
    *                 otherwise its the index
    *
    *.....
    */
    static find_element_id(ID id, BYTE ver)
    {
        int k;
        int ret = -1;
        for(k = 0; k < MAXELEMENTS; k++)
        {
            if(Elements[k].in_use == 'Y' &&
                CCompB(&Elements[k].e_id, &id, sizeof(ID)) == -1 &&
                Elements[k].e_ver == ver)
            {
                ret = k;
                break;
            }
        }
    }
}

```

- 3 byte
- 1 byte version

```
return ret;
```

```
/*.....
```

```
hextoi(int c)
```

```
Description:
```

```
Converts an ASCII hex character into an integer.
```

```
Input:
```

```
c
```

```
character
```

```
- ASCII
```

```
Output:
```

```
Nothing
```

```
Returns:
```

```
-1 if not hex character  
otherwise its the integer equivalent
```

```
/*.....
```

```
static int hextoi(  
int c)
```

```
{  
    char digit, lower, upper;  
    digit = (c >= '0' && c <= '9');  
    lower = (c >= 'a' && c <= 'f');  
    upper = (c >= 'A' && c <= 'F');
```

```
    if (digit)  
        return (c - '0');
```

```
    if (lower)  
        return (c - 'a' + 10);
```

```
    if (upper)  
        return (c - 'A' + 10);
```

```
    return (-1);  
}
```

```
/*.....
```

```
pack_mac_addr(BYTE *packed_address,  
int packed_address_len,  
BYTE *address,  
int address_len)
```

```
Description:
```

```
Converts a character string containing the mac address i  
nto  
packed BCD digits.
```

```
Input:
```

```
packed_address_len  
address
```

```
- length of the packed buffer  
- Hex ASCII string to co
```

```
invert
```

```
I string
```

```
address_len
```

```
- length of the hex ASCII
```

```
Output:
```

```
packed_buffer
```

```
- buffer to write packed address
```

```
into.
```

```
/*.....
```

```
Returns:
```

```
TRUE if packed successfully
```

```
/*.....
```

```
#define LEFT 0
```

```
#define RIGHT 1
```

```
int pack_mac_addr( BYTE *packed_address, int packed_address_len,  
BYTE *address, int address_len)
```

```
{  
    BYTE c, side;
```

```
    int i, j;
```

```
    /*
```

```
    * Pack hex digit ascii string in "address" into binary in  
    * "packed_address". Return FALSE if unsuccessful. If number of hex  
    * digits in "address" cannot fill "packed_address", then  
    * "packed_address" is padded to the right with zeros.  
    */
```

```
    side = LEFT;
```

```
    for (i = 0; i < packed_address_len; i++)  
        packed_address[i] = 0;
```

```
    for (i = 0, j = 0; i < address_len; i++)
```

```
    {  
        if ((c = hextoi(address[i])) == 0xff)  
            return (FALSE);
```

```
        if (side == LEFT)
```

```
        {  
            c = c << 4;
```

```
            packed_address[j] |= c;
```

```
            if (++side > RIGHT)
```

```
            {  
                side = LEFT;  
                if (++j > packed_address_len)  
                    return (FALSE);  
            }  
        }
```

```
    return (TRUE);  
}
```

```
/*.....  
    check_df_groups(void)
```

```
Description:
```

```
This function is called when new PACAU or DACAU is proce  
ssing.  
It checks if there are any changes in groups membership  
to avoid receiving DF elements when membership of this a  
to the proper DF group has been deleted.
```

```
Input:
```

```
Nothing
```

```
Output:
```

```
Nothing
```

```
Returns:
```

```
0 if check was successful
```

```
static check_df_groups(void)
```

```
{
    int i;
```

```
    MUXpacau_t *pacau;
    MUXdacau_t *dacau;
    MUXpeb_t *p_peb;
```

```
    for(i = 0; i < MAXELEMENTS; i++) {
        if(Elements[i].in_use == 'N' || Elements[i].packfeed != 'P')
            continue;
```

```
        if((p_peb = find_peb(Elements[i].e_id, -1)) == NULL)
            continue;
```

```
        pacau = find_pacau(p_peb->groupid, p_peb->version);
        dacau = find_dacau(p_peb->groupid, p_peb->version);
```

```
        if(dacau == NULL && pacau == NULL) {
            /* We have no group for this element */
            del_peb_element(i);
        }
```

```
        return 0;
    }
```

```
/*.....*/
```

```
    parse_pacau(BYTE *buf, WORD len)
```

```
    Description:
```

```
        Parse the PACAU packet. This is where we detect that we
        must receive a new key via the modem(packets d version
        will not match CASDBdacau.version).
```

```
    Input:    buf
```

```
              - pointer to the message receive
```

```
    len
```

```
              - length of the message received
```

```
    Output:  Nothing
```

```
    Returns: 0 if successfully parsed
```

```
/*.....*/
```

```
static parse_pacau(BYTE *buf, WORD len)
```

```
{
    LONG curr_p_version;
```

```
    LONG curr_d_version;
```

```
    LONG curr_d_time;
```

```
    int ret = 0, k;
```

```
    if(strlen(buf, SiteID, 8) != ESUCCESS) {
```

```
        strncpy(SiteID, buf, 8);
```

```
        SiteID[8] = 0;
```

```
        CDBVersion ++;
```

```
        /* New Site ID - reset versions of PACAU, DACAU, ....*/
```

```
        CASDBpacau.version = CASDBpeb.version = 0;
```

```
        CASDBdacau.version = CASDBbecau.version = 0;
```

```
        SaveConfig();
```

```
    }
```

```
    curr_p_version =
```

```
    buf[8] +
```

```
    buf[9] * 256UL +
```

```
    buf[10] * 256UL * 256UL +
```

```
    buf[11] * 256UL * 256UL * 256UL;
```

```
    curr_d_version =
```

```
    buf[12] * 256UL +
```

```
    buf[13] * 256UL * 256UL +
```

```
    buf[14] * 256UL * 256UL * 256UL +
```

```
    buf[15] * 256UL * 256UL * 256UL * 256UL;
```

```
    curr_d_time =
```

```
    buf[16] * 256UL +
```

```
    buf[17] * 256UL * 256UL +
```

```
    buf[18] * 256UL * 256UL * 256UL +
```

```
    buf[19] * 256UL * 256UL * 256UL * 256UL;
```

```
    if(curr_d_version != CASDBdacau.version) {
```

```
        /* There are some changes in the DACAU staff */
```

```
        /* Build DACAU request */
```

```
        DcauFlag = 1;
```

```
        srand(time(0));
```

```
        DcauTime = curr_d_time;
```

```
        RndDcauTime = time(0) + rand();
```

```
        CDBVersion++;
```

```
        WaitingForKeys = TRUE;
```

```
        UpdateFileStatus(MSG("Obtaining Encryption Keys", 138));
```

```
        memcpy(&DcauRequest.d_version, buf + 8 + sizeof(VER), sizeof(VER));
```

```
        #ifdef USE_NEW_MIPS_CODE
```

```
        DcauRequest.d_version.i[3] |= 0x80;
```

```
        #endif
```

```
        CASDBdacau.version = curr_d_version;
```

```
        SaveConfig();
```

```
    }
```

```
    if(curr_p_version != CASDBpacau.version) {
```

```
        CDBVersion++;
```

```
        CASDBpacau.version = curr_p_version;
```

```
        CASDBpacau.length = len - PACAU_HEAD_LEN;
```

```
        memcpy(CASDBpacau.p_buffer,
```

```
            buf + PACAU_HEAD_LEN,
```

```
            CASDBpacau.length);
```

```
        CASDBpacau.entries = CASDBpacau.length / CASDBpacau.entry_len;
```

```
        check_df_groups();
```

```
        SaveConfig();
```

```
    }
    return ret;
```

```
/*.....*/
```

```
    parse_eacu(BYTE *buf, WORD len)
```

```
    Description:
```

```
        Parse the ECAU packet. Replace the old entry by the
        new one as long as version doesn't match CASDBbecau vers
```

```
on.
```

```
    Input:    buf
```

```
              - pointer to the message receive
```

```
    len
```

```
              - length of the message received
```

```
    }
```

```

Output: Nothing

Returns: 0 if successfully parsed

```

```

static parse_ecau(BYTE *buf, WORD len)

```

```

    LONG curr_e_version;
    int ret = 0;

```

```

    curr_e_version =

```

```

        buf[0] +
        buf[1] * 256UL +
        buf[2] * 256UL * 256UL +
        buf[3] * 256UL * 256UL * 256UL;

```

```

    if (curr_e_version != CASDBecau.version)
    {

```

```

        CDBVersion++;

```

```

        CASDBecau.version = curr_e_version;

```

```

        CMovB(buf + ECAU_HEAD_LEN, CASDBecau.p_buffer, len - ECAU_HEAD_LEN);
        CASDBecau.length = len - ECAU_HEAD_LEN;

```

```

        CASDBecau.entries = CASDBecau.length / CASDBecau.entry_len;

```

```

        SaveConfig();
    }

```

```

    return ret;
}

```

```

/*****

```

```

    parse_peb(BYTE *buf, WORD len)

```

```

    Description:

```

```

        Parse the PEB packet. Replace the old entry by the
        new one. Check elements and add new addresses to the
        MLID and delete old ones.

```

```

    Input:

```

```

        buf          - pointer to the message receive
        len          - length of the message received

```

```

    Output: Nothing

```

```

    Returns: 0 if successfully parsed

```

```

static parse_peb(BYTE *buf, WORD len)

```

```

    int i, k, macs, not_found;

```

```

    MUXpeb_t *p_peb;

```

```

    unsigned long curr_version;

```

```

    int ret = 0;

```

```

    MUXpacau_t *pacau;

```

```

    MUXdacau_t *dacau;

```

```

    curr_version =

```

```

        buf[0] +
        buf[1] * 256UL +
        buf[2] * 256UL * 256UL +

```

```

    buf[3] * 256UL * 256UL * 256UL;

```

```

    if (curr_version != CASDBpeb.version)
    {

```

```

        CDBVersion++;

```

```

        CDBethVersion++;

```

```

        CASDBpeb.version = curr_version;

```

```

        CMovB(buf + PEB_HEAD_LEN, CASDBpeb.p_buffer, len - PEB_HEAD_LEN);

```

```

        CASDBpeb.length = len - PEB_HEAD_LEN;

```

```

        CASDBpeb.entries = CASDBpeb.length / CASDBpeb.entry_len;

```

```

        /* Walk throught the elements table and check ... */

```

```

        /* ... if the element still in the PEB */

```

```

        for (i = 0; i < MAXELEMENTS; i++)
        {

```

```

            if (Elements[i].in_use == 'N' || Elements[i].packfeed != 'F')
                continue;

```

```

            if ((p_peb = find_peb(Elements[i].e_id, -1)) == NULL)
            {

```

```

                /* Element no longer valid: Delete. */

```

```

                del_peb_element(i);

```

```

                continue;

```

```

            }

```

```

            if (p_peb->version != Elements[i].e_ver)
            {

```

```

                /* We have found element but with different version: */

```

```

                /* It means, that we somehow miss key update */

```

```

                /* Replace inside adapter and in CDB. */

```

```

                /* Delete address */

```

```

            }

```

```

        }

```

```

        DIODEleteAddress(Elements[i].channel, (BYTE *)&E

```

```

        lements[i].e_mac);

```

```

        /* Replace element in the CDB */

```

```

        replace_peb_element(i, p_peb->version);

```

```

        /* Trying to Add address */

```

```

        /* At first find group for the element */

```

```

        pacau = find_pacau(p_peb->groupid, p_peb->version);

```

```

        dacau = find_dacau(p_peb->groupid, p_peb->version)

```

```

        n);

```

```

        if (dacau != NULL)

```

```

        {

```

```

            pacau = (MUXpacau_t *)dacau;

```

```

            if (pacau != NULL)
            {

```

```

                if (DIOAddGroupAddress(Elements[i].chann

```

```

                el, (BYTE *)&Elements[i].e_mac,

```

```

                /* Find a group. Add */

```

```

                channelCfg.CfgChannel = Elements[i].chan

```

```

                channelCfg.CfgNumAddresses = 1;

```

```

                CMovB(&Elements[i].e_mac, channelCfg.Cfg

```

```

                CMovB(&pacau->g_key, key, 8);

```

```

                reverse_key(&key);

```

```

                CMovB(key, channelCfg.CfgGroupKey, 8);

```

```

                CMovB(&Magickey, channelCfg.CfgElementKe

```

```

                y, 8);

```

```

                ecb.ECB_StackID = MLID_ADD_ADDRESS;

```

```

                ecb.ECB_Fragment[0].FragmentAddress = &c

```

```

                if (toctmId(FDBBoard, &ech, FDBControl

```

```

                del_peb_element(i);

```

```

            }

```

```

        }

```

```

    }

```

```

    hannelCfg;

```

```

    Entry) != 0)

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    else
    (
        /* Can't find group for this element */
        /* We are not subscribed on this group any more */
        /* Delete element. */
        del_peb_element(i);
    )

    /* Check Ethernet addresses within Element. */
    macs = not_found = 0;
    for(k = 0; k < MAX_MAC_RECORDS; k++)
    (
        if(CASmacs[k].in_use == 'Y' &&
           CCompB(&CASmacs[k].dpc_mac, &Elements[i].e_mac, 8
              sizeof(MACaddr_t)) == -1)
        (
            macs++;
            if(!find_peb_mac(CASmacs[k].e_mac) == NULL)
            (
                /* Somebody in the NOC has deleted Ethernet */
                /* address of this element... */
                not_found++;
                CASmacs[k].in_use = 'N';
            )
        )
    )

    if(macs == not_found)
    (
        /* There are no ethernet addresses for the element */
        del_peb_element(i);
    )

    SaveConfig();
    return ret;
}

/*****
 *
 * parse_gup(BYTE *buf, WORD len)
 *
 * Description:   Parse the GUP packet. Add new entries to the PACAU buffer.
 *
 * Input:        buf
 *               - pointer to the message receive
 *               len
 *               - length of the message received
 *
 * Output:       Nothing
 *
 * Returns:      0 if successfully parsed
 *
 * *****/
static parse_gup(BYTE *buf, WORD len)
(
    GUPid_t *p_gup_element;
    GUPhead_t *p_gup_head;
    MUXpacau_t *p_pacau_tmp;
    int i, curr_len = 0;

```

```

    int index, start, end;

    p_gup_head = (GUPhead_t *)buf;
    index = CCompB(&p_gup_head->adapterstart, SerialNum, sizeof(ID));
    if (index == -1)
        start = 0;
    else
        start = p_gup_head->adapterstart.i[index] - SerialNum[i];

    index = CCompB(&p_gup_head->adapterend, SerialNum, sizeof(ID));
    if (index == -1)
        end = 0;
    else
        end = p_gup_head->adapterend.i[index] - SerialNum[i];

    if(start <= 0 && end >= 0)
    (
        CDBVersion++;
        for(i = 0; i < p_gup_head->entries && curr_len < len; i++, len += GUP_LEN)
        (
            p_gup_element = (GUPid_t *) (buf + GUP_HEAD_LEN + i * GUP_LEN);
            if(CCompB(&p_gup_element->adapternum, SerialNum, sizeof(ID)) == -1)
            (
                p_pacau_tmp = (MUXpacau_t *) (CASDBpacau.p_buffer + CASDBpacau.length);
                CASDBpacau.length += PACAU_LEN;
                CASDBpacau.entries++;
                CMovB(&p_gup_head->groupid, &p_pacau_tmp->groupid, sizeof(ID));
                p_pacau_tmp->version = p_gup_head->g_ver;
                CMovB(&p_gup_element->g_key, &p_pacau_tmp->g_key, sizeof(chunk));
                break;
            )
        )
        return 0;
    )

/*****
 *
 * update_peb(BYTE *buf, WORD len)
 *
 * Description:   Parse the UPDATE_PEB packet. Replace the old element with
 *               new one according to the Element ID.
 *
 * Input:        buf
 *               - pointer to the message receive
 *               len
 *               - length of the message received
 *
 * Output:       Nothing
 *
 * Returns:      0 if successfully parsed
 *
 * *****/
static update_peb(BYTE *buf, WORD len)
(
    int m;
    MUXpacau_t *pacau = NULL;
    MUXdcau_t *dcau = NULL;

```

```

MUXpeb_t *old_peb, *new_peb, *found;
int found_element;
short ret_code = 0;
unsigned long curr_version;

```

```

if(len != 2 * PEB_LEN + PEB_HEAD_LEN)
    return(0);
curr_version =

```

```

    buf[0] +
    buf[1] * 256UL +
    buf[2] * 256UL * 256UL +
    buf[3] * 256UL * 256UL * 256UL;

```

```

if(curr_version == CASDBpeb.version)
    return 0;
CDBethVersion++;
CASDBpeb.version = curr_version;

```

```

CDBethVersion++;

```

```

CASDBpeb.version = curr_version;

```

```

old_peb = (MUXpeb_t *) (buf + PEB_HEAD_LEN);
new_peb = (MUXpeb_t *) (buf + PEB_HEAD_LEN + PEB_LEN);
if((found = find_peb(old_peb->elementid, old_peb->version)) == NULL)
    /* Nothing to replace - ERROR */
    return(0);

```

```

for(m = 0; m < MAXELEMENTS; m++)

```

```

{
    if(UpdatedElements[m].in_use == 'Y')
    {

```

```

        /* We have received next replace element command,

```

```

        * it means, that the previous element

```

```

        * has already been updated at the

```

```

        * DataFeed Lan Gateway and

```

```

        * we can delete old address and keys

```

```

        * from adapter
        */
    }
}

```

```

ret_code = DIDeleteAddress(UpdatedElements[m].channel,
    (BYTE *) &UpdatedElements[m].e_mac);

```

```

UpdatedElements[m].in_use = 'N';

```

```

/* Has been the element loaded before? */
found_element = find_element_id(old_peb->elementid, old_peb->version);
if(found_element != -1)
{

```

```

    /* Yes. We are receiving this element now.

```

```

    * Let's keep old element's address

```

```

    * in the UpdatedElement table
    */
CMovB(&Elements[found_element], &UpdatedElements[found_element],
    sizeof(CDBelement_t));

```

```

/* Trying to find a group for the element
*/
pacau = find_pacau(new_peb->groupid, new_peb->grversion);
dacau = find_dacau(new_peb->groupid, new_peb->grversion);
if(dacau != NULL)
    pacau = (MUXpacau_t *) dacau;
    if(pacau != NULL)
    {

```

```

        /* Put element in the adapter

```

```

        * After this operation in the adapter will be

```

```

        * both old and new element's addresses and keys
        */
    }
}

```

```

/* copy the data past the lroinfo to the message */
CMovB(ecb->ECB_Fragment[0].FragmentAddress, message, ecb->ECB_Fragment[0].
    FragmentLength);

```

```

/* return the ecb to the LSL */

```

```

return the ecb to the LSL */

```

```

delay(120);

```

```

ret_code = DIAddGroupAddress(Elements[found_element].ch
    (BYTE *) &pacau->g_key);
}
else
    /* No group, Delete element */
    del_peb_element(found_element);
}
/* Change PEB database for this element */
if(!ret_code)
{
    replace_peb_element(found_element, new_peb->version);
    CMovB( (unsigned char *) new_peb, (unsigned char *) found, PEB_LEN - 2);
    return(ret_code);
}
/******
AccessReceive(char *message)
Description:
This routine checks to see if we've received any
packets from the MUID. If we have, the data is copied
from the ECB to the message and the ECB is returned to t
he
LSL.
Input:
message
r to where to copy data
Output:
message and lroinfo filled in if successful
Returns:
0 if a packet has been received
*****
LONG AccessReceive(char *message)
ECB *ecb;
/* Extract an ECB from the linked list if one exists */
Disable();
ecb = AccessECBHead;
if (!ecb)
{
    /* No ecb. Just return */
    Enable();
    return(-1);
}
AccessECBHead = ecb->ECB_NextLink;
if (AccessECBHead == 0)
    AccessECBTail = 0;
Enable();
/* copy the data past the lroinfo to the message */
CMovB(ecb->ECB_Fragment[0].FragmentAddress, message, ecb->ECB_Fragment[0].
    FragmentLength);
/* return the ecb to the LSL */

```

```

    CLSRReturnRCVECB(ecb);
#endif LOG_ECB_ACTIVITY
    FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
        "ACCESS return(%08lx)\n", ecb));
#endif /* LOG_ECB_ACTIVITY */
    return(0);
}

/*****
 *
 * AccessAdd(BYTE *message)
 *
 * Description: This routine is called when a packet is received
 *              and is responsible for dispatching it.
 *
 * Input:      message
 *             lroinfo
 *
 * Output:     Nothing
 *
 * Returns:    Nothing
 *
 *****/

int AccessAdd(BYTE *message) {
    IndPacket_t *p_packet;
    int packet_type;
    int status;

    p_packet = (IndPacket_t *)message;

    if (p_packet->address[3] == 0x01)
        switch(p_packet->payload(0)) {
            case SG_PACAU:
                packet_type = PACAU;
                break;
            case SG_ECAU:
                packet_type = ECAU;
                break;
        }
    else if (p_packet->address[0] == 0x03) { /* Bypass key */
        switch(p_packet->payload(0)) {
            case PEB_PACKET:
                packet_type = PEB;
                break;
            case GUP_PACKET:
                packet_type = GUP;
                break;
            case PEB_UPDATE_PACKET:
                packet_type = PEB_UPDATE;
                break;
            default:
                packet_type = UNKNOWN;
                break;
        }
    }
    else
        packet_type = UNKNOWN;

    switch(packet_type) {
        case UNKNOWN:
            break;
    }
}

```

```

case PACAU:
    status = parse_pacau(p_packet->payload+1, p_packet->length-1);
    break;
case ECAU:
    status = parse_ecau(p_packet->payload+1, p_packet->length-1);
    break;
case PEB:
    status = parse_peb(p_packet->payload+1, p_packet->length-1);
    break;
case GUP:
    status = parse_gup(p_packet->payload+1, p_packet->length-1);
    break;
case PEB_UPDATE:
    status = update_peb(p_packet->payload+1, p_packet->length-1);
    break;
}
return(status);
}

/*****
 *
 * parse_dacau(BYTE *buf,
 *             int len)
 *
 * Description: Parse the new DACAU from the message received by the mod
 *
 *              Replace the old DACAU buffer by the new one if the v
 *              matches what we have in the CASDBdacau version.
 *
 * Input:      buf
 *             len
 *
 * Output:     Nothing
 *
 * Returns:    0 if successful
 *
 *****/

static int parse_dacau(BYTE *buf, int len)
{
    unsigned long curr_d_version;
    int ret = 0;

    curr_d_version =
        buf[0] +
        buf[1] * 256UL +
        buf[2] * 256UL * 256UL +
        buf[3] * 256UL * 256UL * 256UL;

    if (curr_d_version == CASDBdacau.version) {
        CDBVersion++;
        CASDBdacau.version = curr_d_version;
        CMovB(buf + DACAU_HEAD_LEN, CASDBdacau.p_buffer, len - DACAU_HEAD_LEN);
        CASDBdacau.length = len - DACAU_HEAD_LEN;
        CASDBdacau.entries = CASDBdacau.length / CASDBdacau.entry_len;
        check_df_groups();
    }
    else {
        ret = -1;
    }
}

```

```
return ret;
}

/*****
 *
 * DacauResponse(BYTE *pdata,
 *              int len,
 *              timeoutFlag)
 *
 * Description:      This is the callback routine pass in to DloScheduleRecei
 *                   which receives the response from the modem. If a timeout
 *                   hadn't occurred, parse the message for the dacau info.
 *
 * Input:      pdata
 *              len
 *              timeoutFlag
 *
 * ssage
 * received
 *
 *              - non-zero if we timed out
 *
 * Output:      Nothing
 *
 * Returns:      Nothing
 *
 * *****/
static void DacauResponse(BYTE *pdata, int len, int timeoutFlag)
{
    if (timeoutFlag)
    {
        DacauFlag = 1;
        UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 258));
        return;
    }
    EnterDebugger();
    if (parse_dacau(pdata, len) == 0)
    {
        WaitingForKeys = FALSE;
        CDBVersion++;
        SaveConfig();
        UpdateFileStatus(MSG("IDLE", 211));
    }
    else
    {
        DacauFlag = 1;
        UpdateFileStatus(MSG("Retry: Obtaining Encryption Keys", 259));
    }
}

/*****
 *
 * GetDacau(void)
 *
 * Description:      Initiate the reception of the encryption keys for this a
 *                   by constructing a request, sending it to the modem, and
 *                   scheduling a receive routine to receive the response whi
 *                   hopefullly contains our encryption key.
 *
 * Input:
 *
 * *****/
```

```
Nothing
Output:      Nothing
Returns:      Nothing
*****/
static void GetDacau(void)
{
    int k;
    BYTE c;

    memcpy(&DacauRequest.adapterNum, SerialNumPacked, sizeof(ID));
    if (DIOSignText( (BYTE *)&DacauRequest, sizeof (DacauRequest_t), Dac
auRequest.sign) != 0)
        return;

    for(k = 0; k < 8; k+=2)
    {
        c = DacauRequest.sign[k];
        DacauRequest.sign[k] = DacauRequest.sign[k+1];
        DacauRequest.sign[k+1] = c;
    }
    SlipSend((BYTE *)&DacauRequest, sizeof(DacauRequest_t), 1, DLO_GETKEYS_T
IMEOUT);
    if (DloScheduleReceive(DacauResponse, 60, DACAU_RECEIVE) == 0)
        DacauFlag = 0;
}

/*****
 *
 * AccessMain(void *parm)
 *
 * Description:      Main access thread which handles conditional access
 *                   packet reception. We'll start out by reading DPC.CFG.
 *                   If the serial number on the adapter is different from th
 *                   in DPC.CFG, we'll call out immediately to get the ke
 *                   Then we'll just sleep until there is a packet to handle.
 *
 * Input:      parm
 *
 * Output:      Nothing
 *
 * Returns:      Nothing
 *
 * *****/
void AccessMain(void *parm)
{
    LONG sn, ccode;
    ECB ecb = {0, 0, 0, 0, 0, 0, MSG("DRCTPC", 212)};
    ChannelConfig channelCfg;
    BYTE address[8];
    BYTE x;
    LONG removedCount;
}
```

```

parm = parm;

/*
 * When MLID has been found, Open the conditional access channel.
 */

RegisterWithDriver:

while(!ExitingFlag)
(
    AccessAsleep = TRUE;
    Delay(500);
    AccessAsleep = FALSE;

    if (DIOBoard != 0)
    (
        removedCount = DIORemovedCount;
        if (AccessChannel != -1)
            DIOOpenChannel(AccessAddress, AccessESR,
                &AccessChannel) == 0)
        (
            DIOGetSN(SerialNum);
            sn = atol(SerialNum);
            NWaprintf(SerialNum, MSG("%06lX", 213), sn);

            pack_mac_addr(SerialNumPacked, 3, SerialNum, 6);
            x = SerialNumPacked(0);
            SerialNumPacked(0) = SerialNumPacked(2);
            SerialNumPacked(2) = x;

            MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACAddr
                _t *)address);

            if (DIOAddrAddress(AccessChannel, address) == 0)

                channelCfg.CfgChannel = AccessChannel;
                channelCfg.CfgNumAddresses = 1;
                MACbuildAddr(SerialNum, MAC_CAS_IND, 0, (MACAddr
                    _t *)channelCfg.CfgAddress);

                ecb.ECB_StackID = MLID_ADD_ADDRESS;
                ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;

                if (IoctlMlId(FDBBoard, &ecb, FDBControlEntry) =
                    = 0)
                    break;

            )
        )

    if (ExitingFlag)
    (
        DPCAccessPID = 0;
        return;
    )

/*
 * Now lets open up the the DPC.CFG file.
 */

ReadConfig();

while(!ExitingFlag)
(

```

```

ccode = AccessReceive(AccessMessage);
if (ccode)
(

```

```

    AccessAsleep = TRUE;
    SuspendThread(DPCAccessPID);
    AccessAsleep = FALSE;
    if (removedCount != DIORemovedCount)
        goto RegisterWithDriver;
    continue;
)

```

```

    AccessAdd(AccessMessage);
    if (DcauFlag)
        GetDcau();
)

```

```

    DIOCloseChannel(AccessChannel);
    DPCAccessPID = 0;
}

```

```

#include "dpcagent.h" /* Our header file */

LONG DIOBoard = 0;
LONG DIORemovedCount = 0;
LONG DIOControlEntry = 0;
struct DriverStatsStructure* DIOStats = 0;

void DIORemove(void)
{
    int i;

    /* Invalidate the DriverIO board. Increment removed count so that
     * other threads can detect that the driver has changed, even if
     * DIOBoard gets filled in. The other threads can then re-register
     * with the new driver.
     */
    DIOStats = 0;
    DIOBoard = 0;
    DIORemovedCount++;

    /* Force the NMSNUT menus to exit so that DPCAGENT can go back
     * to searching for a new adapter before allowing user to choose
     * options.
     */
    for(i = 0; i < 4; i++)
        NMSUngetKey(UGK_ESCAPE_KEY, UGK_NORMAL_KEY, NUTHandle);

    /* Force sleeping threads to wake up so that they can detect that
     * the adapter has been unloaded(DIORemovedCount will be different
     * from their local copy of the last removed count). The threads
     * should then spin(sleep) periodically until a new adapter is
     * present, and then re-register with the adapter if they need to.
     */

    if (AccessAsleep)
        ResumeThread(DPCAccessPID);

    if (InetAsleep)
        ResumeThread(DPCInetPID);

    LONG board;

    LONG DIORegisterWithAdapter(char *shortName)
    {
        for(board = 0; board < NumberOfLANs; board++)
        {
            void (*ControlEntryPoint) (void) = NULL;
            if (CLSLGetMLIDControlEntry(board,
                                         &ControlEntryPoint) == ESUCCESS)
            {
                struct DriverConfigurationStructure* config = 0;
                if (!config = (struct DriverConfigurationStructure *) Co
                    mmandMLID(board, 0, (LONG)ControlEntryPoint)))
                {
                    if (!CStrCmp(config->DShortName, shortName))
                    {
                        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC",
505));

```

```

void (*removeRoutine) () = DIORemove;

ecb.ECB_StackID = MLID_REGISTER_AGENT;
ecb.ECB_Fragment[0].FragmentAddress = &r

/* Call MLID */
IoctlMLID(board, &ecb, (LONG)ControlEntr

DIOBoard = board;
DIOControlEntry = (LONG)ControlEntryPoin
return(0);

        )
        return(-1);
    )

void DIODeRegisterAgent(void)
{
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 506));
    void (*nullRoutine) () = 0;
    if (DIOBoard)
    {
        ecb.ECB_StackID = MLID_REGISTER_AGENT;
        ecb.ECB_Fragment[0].FragmentAddress = &nullRoutine;

        /* Call MLID */
        IoctlMLID(DIOBoard, &ecb, DIOControlEntry);
        DIOBoard = 0;
    }

    /*.....*/

    DIOGetSN(char *serialNum)

    Description:
        This routine Gets the serial number from the adapter.
        It is used for explicit requests of packages that are
        for sale.

    Input:
        serialNum
        al number

    Output:
        serialNum
        - filled out if successful

    Returns:
        0 if successful
        /*.....*/

    LONG DIOGetSN(char *serialNum)
    {
        LONG ccode;
        ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 108));
        if (!DIOBoard)
            return(-1);

```

```
ecb.ECB_StackID = MLID_GET_SN;
ecb.ECB_Fragment(0).FragmentAddress = serialNum;

/* Call MLID */
ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
return(ccode);
)
```

```
/*.....
 * DIOSignText(char *textToSign,
 *             LONG textLength,
 *             char *signature)
 *.....
```

Description:

This routine uses the adapter to calculate a signature for the given text. It is used for explicit requests of packages that are for sale.

Input:

textToSign - string to be signed
textLength - length of string to be signed
signature - string to store signature

Output:

signature - filled out if successful

Returns:

0 if successful

```
LONG DIOSignText(char *textToSign,
                 LONG textLength,
                 char *signature)
{
    LONG ccode;
    LONG fragment(3);
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 139)};
    if (!DIOBoard)
        return(-1);
    fragment(0) = (LONG)textToSign;
    fragment(1) = textLength;
    fragment(2) = (LONG)signature;
    ecb.ECB_StackID = MLID_SIGN_TEXT;
    ecb.ECB_Fragment(0).FragmentAddress = fragment;
    /* Call MLID */
    ccode = IoctlMlid(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}
```

EXPORTED FUNCTION

DPCGetMLIDStats(struct DriverStatsStructure **stats)

Description: This routine fills in a pointer to the MLID stats table.

Input: stats
 ter to stats table

Output: stats filled in if successful

Returns: 0 if MLID is active

```
int DPCGetMLIDStats(struct DriverStatsStructure **stats)
{
    if (!DIOBoard)
        return(-1);
    *stats = DIOStats = (struct DriverStatsStructure *)
        CommandMlid(DIOBoard, 1, DIOControlEntry);
    return(0);
}
```

```
int DIOGetMLIDConfig(struct DriverConfigurationStructure **config)
{
    if (!DIOBoard)
        return(-1);
    *config = (struct DriverConfigurationStructure *)
        CommandMlid(DIOBoard, 0, DIOControlEntry);
    return(0);
}
```

```
DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)
```

Description:

This routine attempts to open an adapter channel for the passed in bypass address, such as 0f 00 00 00 00.

Input: address
 address for open

esr
- ESR address for this channel
channel
channel number is returned

Output: channel is filled out if successful

Returns: 0 if channel was opened

```
LONG DIOOpenChannel(BYTE *address, int (*esr)(), LONG *channel)
{
    ChannelConfig cfg;
    - Bypass
    - where
```



```

    if (!DIOBoard)
        return(-1);

    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;
    CMovB(address, channelCfg.CfgAddress, 8);
    CMovB(groupAddress, key, 8);
    reverse_key((BYTE*)&key);
    CMovB(key, channelCfg.CfgGroupKey, 8);
    CMovB(&MagicKey, channelCfg.CfgElementKey, 8);

    ecb.ECB_StackID = MLID_ADD_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    ccode = IoctlMlId(DIOBoard, &ecb, DIOControlEntry);
    return(ccode);
}

int DIOAddHIAddr(unsigned char channel, BYTE *hiAddr)
{
    chunk key;
    ID hi_id;
    int i, ret = CAS_ERROR;
    ChannelConfig channelCfg;
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 471)};

    if (!DIOBoard)
        return(-1);

    for(i = 0; i < 3; i++)
        hi_id[i] = 0x00;
    make_hi_key(&key);
    channelCfg.CfgChannel = channel;
    channelCfg.CfgNumAddresses = 1;

    CMovB(hiAddr, channelCfg.CfgAddress, 8);
    reverse_key((BYTE*)&key);
    CMovB(&key, channelCfg.CfgGroupKey, 8);
    CMovB(&key, channelCfg.CfgElementKey, 8);

    channelCfg.CfgChannel = FDBChannel;
    channelCfg.CfgESR = FDB_ESR;
    channelCfg.CfgNumAddresses = 1;
    CMovB(&addr, channelCfg.CfgAddress, 8);
    CMovB(&pacau->g_key, key, 8);
    reverse_key(&key);
    CMovB(key, channelCfg.CfgGroupKey, 8);
    CMovB(&MagicKey, channelCfg.CfgElementKey, 8);

    channelCfg.CfgESR = (int (*)())0xffffffff;
    ecb.ECB_StackID = MLID_ADD_ADDRESS;
    ecb.ECB_Fragment[0].FragmentAddress = &channelCfg;
    CMovB(&addr, node->file_address, 6);

    ret = IoctlMlId(DIOBoard, &ecb, DIOControlEntry);

    if((ret == WBICDDAddAddress
        ((BICDD_CHANNEL_CONFIG FAR *)&channel_config))!=CAS_OK)
        return ret;
}

int
DIORegisterSend(int (*sendRoutine)(TCB *))
{
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 481)};

    if (!DIOControlEntry)
        return(-1);

    ecb.ECB_StackID = MLID_REGISTER_SEND_ROUTINE;
    ecb.ECB_Fragment[0].FragmentAddress = &sendRoutine;

    /* Call MLID */
    IoctlMlId(DIOBoard, &ecb, DIOControlEntry);
    return(0);
}

int
DIOObtainReturnTCB(void (**returnTCBRoutine)(TCB *))
{
    ECB ecb = {0, 0, 0, 0, 0, MSG("DRCTPC", 147)};

    if (!DIOControlEntry)
        return(-1);

    ecb.ECB_StackID = MLID_RETURN_TCB_ROUTINE;
    ecb.ECB_Fragment[0].FragmentAddress = returnTCBRoutine;

    /* Call MLID */
    IoctlMlId(DIOBoard, &ecb, DIOControlEntry);
    return(0);
}

int
DPCGetIPAddress(LONG* ip) {
    LONG id;
    LONG (*ctl)(LONG board, ...);
    char buf[80];
    char* s = buf;

    if (CLSLGetStackIDFromName("\002IP", &id))
        return 1;
    if (CLSLGetProtocolControlEntry(id, DIOBoard, &ctl))
        return 2;
    if (GetProtocolStringForBoard(ctl, DIOBoard, buf))
        return 3;
    s = strpbrk(buf, "0123456789");
    if (!s)
        return 4;
    *ip = inet_addr(s);
    if (*ip == (LONG)(-1))
        return 5;
    return 0;
}

```



```

int handle;

DloCfg.ip_address = ntohl(DloCfg.ip_address);
DloCfg.gateway_address = ntohl(DloCfg.gateway_address);
handle = open(MSG("SYS:DIREPCPC\\DB\\MODEM.CFG", 335),
              O_RDWR | O_CREAT,
              S_IWWRITE | S_IREAD);

if (handle != -1)
{
    write(handle, &DloCfg, sizeof(DloCfg));
    close(handle);
}

DloCfg.ip_address = htonl(DloCfg.ip_address);
DloCfg.gateway_address = htonl(DloCfg.gateway_address);
}

#define SerialWarn(s) sprintf(warnbuf, "\r\nDPCN: detected a bad serial number: %8.8s\r\n", (char*)(s)), ConsolePrintf(warnbuf), RingTheBell()

static inline unsigned long find_and_clear_low_bit(unsigned long* val) {
    unsigned long low = *val;
    if (low == 0)
        return 0;
    *val &= low - 1;
    return (*val ^ low);
}

static inline int parity(LONG serial) {
    int i;
    for (i = 0; find_and_clear_low_bit(&serial); ++i)
        return (i & 1);
}

static inline int UserCount(BYTE* s) {
    LONG serial = 0;
    int shift;

    s += 3;
    for (shift = 16; shift >= 0; shift -= 4) {
        serial |= (*s - ((*s >= 'A') ? 56 : 0x30)) << shift;
        ++s;
    }
    return 5 * ((serial & 0x00002) >> 1) |
               ((serial & 0x20000) >> 16) |
               ((serial & 0x02000) >> 11) |
               ((serial & 0x00040) >> 3));
}

void DPCSetMaxConnections(LONG* sum) {
    char warnbuf[120];
    int users;
    int pd = 0;
    int i;

    sum[0] = sum[1] = (-1);

    /* re-read modem.cfg file */
    i = DloCfg.ip_address;
    DPCUpdateConfig(i);
    DloCfg.ip_address = i;

```

```

if (strcmp(DloCfg.base_license, "Helius, Inc.", 8) == ESUCCESS) {
    DPCMaxConnections = 108;
    PackageDelivery = 1;
    memcpy(sum, DloCfg.base_license, 2 * sizeof(LONG));
    return;
}

/* check for old license info */
if (atoi(DloCfg.base_license) < 02) {
    sprintf(warnbuf,
            "\r\nDPCN: deactivated old serial number: %8.8s\r\n",
            DloCfg.base_license);
    ConsolePrintf(warnbuf);
    RingTheBell();
    return;
}

/* handle base license first */
memcpy(sum, DloCfg.base_license, 2 * sizeof(LONG));
if (parity(sum[0]) == 0) {
    SerialWarn(DloCfg.base_license);
    return;
}
if (parity(sum[1]) == 0) {
    SerialWarn(DloCfg.base_license);
    return;
}
users = UserCount(DloCfg.base_license);
if (DloCfg.base_license[2] == '*')
    pd = 1;

/* now handle additive licenses */
for (i = 0; i < 9; ++i) {
    int j;
    LONG serial[2];
    if (DloCfg.add_license[i][0] == 0)
        continue;
    /* check for duplicate license */
    for (j = i - 1; j >= 0; --j) {
        if (memcmp(DloCfg.add_license[i],
                  DloCfg.add_license[j],
                  sizeof(DloCfg.add_license[i])) == ESUCCESS) {
            memset(DloCfg.add_license(i), 0, sizeof(DloCfg.add_license(i)));
            DPCUpdateConfigFile();
            sprintf(warnbuf,
                    "\r\nDPCN: deleted duplicate license number: %8.8s\r\n",
                    DloCfg.add_license[j]);
            ConsolePrintf(warnbuf);
            RingTheBell();
            goto nextLicense;
        }
    }
    memcpy(serial, DloCfg.add_license(i), sizeof(serial));
    if (parity(serial[0]) == 1) {
        SerialWarn(DloCfg.add_license(i));
        return;
    }
    if (parity(serial[1]) == 1) {
        SerialWarn(DloCfg.add_license(i));
        return;
    }
    users += UserCount(DloCfg.add_license(i));
    sum[0] += serial[0];
    sum[1] += serial[1];
    if (DloCfg.add_license[i][2] == '*')

```

Thu Jul 17 14:46:12 1997

license.c

Page 5

```
pd = 1;  
nextLicense:  
    ;  
    )
```

```
DPCMaxConnections = users * 4;  
PackageDelivery = pd;  
    )
```

```

#include <pcagent.h> /* Our header file */

#define USE_AIO_DEADMAN 0
#define TRACE_STATE 0

static char* printables = MSG("A..Za..z0..9 -.~!@#$%^&*()_+={}|;:'.<>?\\/\\"
523);
#define dial_chars printables
#define modem_control_chars printables

int AIOPortHandle = -1;
LONG AIOWriteBufferSize = 0;
int AIOGlobalPort = 0;
static BYTE AIOBaudRateDefines[] =
{
    AIO_BAUD_2400, /* 0 */
    AIO_BAUD_3600, /* 1 */
    AIO_BAUD_4800, /* 2 */
    AIO_BAUD_7200, /* 3 */
    AIO_BAUD_9600, /* 4 */
    AIO_BAUD_19200, /* 5 */
    AIO_BAUD_38400, /* 6 */
    AIO_BAUD_57600, /* 7 */
    AIO_BAUD_115200 /* 8 */
};

int DloState = DLOS_IDLE;
LONG DloTimer = 0;
LONG DloPacketLifeTimer = 0;

LONG DloConn = DLO_CONN_IDLE;
LONG DloNextConn = DLO_CONN_IDLE;

LONG DloPxmCount = 0;
LONG DloPxmBufferSize = DLOBUFSIZE;
BYTE DloPxmBuffer[DLOBUFSIZE];
LONG DloPinactivityTimer = 10 * 19;

LONG DloIXmitCount = 0;
LONG DloIXmitBufferSize = DLOBUFSIZE;
BYTE DloIXmitBuffer[DLOBUFSIZE];
LONG DloIXmitactivityTimer = 60 * 19;

LONG DloLastKnownTickCount;
LONG DloRcvCount = 0;
LONG DloRcvIndex = 0;
LONG DloReadIndex = 0;
BYTE DloRcvBuffer[DLOBUFSIZE];
LONG DloCommandIndex = 0;
BYTE DloCommandBuffer[DLOCMDBUFSIZE];
LONG DloLastDCD = 0;

char ConnectBaudStr[DLOCMDBUFSIZE] = (0);

static char *DloCompareString[DLOENUM] =
{
    "",
    Dlocfg.connect_str,
    Dlocfg.disconnect_str,
    "",
    MSG("OK", 219),
    MSG("BUSY", 220),
    MSG("NO DIAL TONE", 221),
    MSG("RING", 222),
    ...
};

);
MSG("NO ANSWER", 99)

/*
 * Variables used by DloScheduleReceive().
 */

void (*DloCallBack)(BYTE *pdata, int len, int timeoutFlag) = 0;
int DloCallBackTimeout = 0;
int DloCallBackWait = 0;
BYTE DloCallBackBuffer[4000];
int DloCallBackIndex = 0;
int DloCallBackStarted = 0;
int DloCallBackEscape = 0;
int DloCallBackType = 0;

/*
 * RS232C NLM Default Init String
 */
//BYTE ModemInitString[] = MSG("ATH0Q0V1X4S0", 72);
/*
 * Digitan DS144FVM, Multitech Auto Reliable and Practical
 * Peripherals V.34 Default Init String
 */
//BYTE ModemInitString[] = "ATE1Q0V1X46C1d2&K0 S7=60 S
11=55";
/*
 * Hayes-Compatible Modem Default Init String
 */
//BYTE ModemInitString[] = MSG("ATE1Q0V1X46C1d2 S7=60
S11=55", 156);
/*
 * Intel 144e Faxmodem and Motorola UDS V.3225
 * Default Init String
 */
//BYTE ModemInitString[] = "ATE1Q0V1X46C1d2\G S7=60 S1
1=55";
/*
 * U.S. Robotics Default Init String
 */
//BYTE ModemInitString[] = MSG("ATE1Q0V1X46C1d2&K0&H0&
10 S7=60 S11=55", 100);

/* internal function prototypes */
void InitializeAIO(void);
void SendAIOData(BYTE *data, LONG length);
static void DloStartTimer( LONG ticks );
//void DloStopTimer( void );
static void DloStopPacketLifeTimer( void );
static void DloStartPacketLifeTimer( void );
static void StateMachine (int event);
static void WriteCommXmitBuffer( void );

void HexAsciiDump(const unsigned char* buffer, unsigned int len)
{
    char display[80];
    while (len >= 16)
    {
        printf(MSG("02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x %02x", 802)
        buffer[0],
        buffer[1],
        ...
    }
}

```

```

buffer[2],
buffer[3],
buffer[4],
buffer[5],
buffer[6],
buffer[7],
buffer[8],
buffer[9],
buffer[10],
buffer[11],
buffer[12],
buffer[13],
buffer[14],
buffer[15],
isprint(buffer[0]) ? buffer[0] : ' ',
isprint(buffer[1]) ? buffer[1] : ' ',
isprint(buffer[2]) ? buffer[2] : ' ',
isprint(buffer[3]) ? buffer[3] : ' ',
isprint(buffer[4]) ? buffer[4] : ' ',
isprint(buffer[5]) ? buffer[5] : ' ',
isprint(buffer[6]) ? buffer[6] : ' ',
isprint(buffer[7]) ? buffer[7] : ' ',
isprint(buffer[8]) ? buffer[8] : ' ',
isprint(buffer[9]) ? buffer[9] : ' ',
isprint(buffer[10]) ? buffer[10] : ' ',
isprint(buffer[11]) ? buffer[11] : ' ',
isprint(buffer[12]) ? buffer[12] : ' ',
isprint(buffer[13]) ? buffer[13] : ' ',
isprint(buffer[14]) ? buffer[14] : ' ',
isprint(buffer[15]) ? buffer[15] : ' ');
buffer += 16;
len -= 16;
}

if (len)
{
/* the basic theory here is to build the buffer in place and the
   insert the extra spaces */
unsigned int n = 0;
register char* d = display;

while (n < len)
{
    NWsprintf(d, MSG("%02x ", 483), buffer[n]);
    d += 3;
    display[(16 * 3 + 2) + n] = isprint(buffer[n]) ? buffer[
        ++n;
    ]
    display[(16 * 3 + 2) + len] = 0;
    memset(d, ' ', (16 - len) * 3 + 2);
    d = display + (16 / 2 * 3);
    memmove(d + 2, d, sizeof(display) - (16 / 2 * 3) - 2);
    d[0] = d[1] = ' ';
    d = display + (16 * 3) + 4 + 8;
    memmove(d + 2, d, 9);
    d[0] = d[1] = ' ';
    puts(display);
}

void DloUpdateModemStr( void )
{
    if (DloState == DLOS_IDLE)
        UpdateModemStr(MSG("Modem Status: IDLE
        \n", 201));
}

```

```

else if (DloState == DLOS_INIT)
    UpdateModemStr(MSG("Modem Status: Initializing Modem
    \n", 240));
else if (DloState == DLOS_DIAL)
    UpdateModemStr(MSG("Modem Status: Dialing
    \n", 236));
else if (DloState == DLOS_REDL)
    UpdateModemStr(MSG("Modem Status: Redialing
    \n", 235));
else if (DloState == DLOS_CONN)
{
    if (DloConn == DLO_CONN_PACKAGE)
        UpdateModemStr(MSG("Modem Status: Connected to Package D
        \n", 241));
    else
    {
        if (ConnectBaudStr[0])
        {
            BYTE connectStr[80];

            NWsprintf(connectStr, MSG("Modem Status: Connect
            ed to Internet at %.24s\n", 473), ConnectBaudStr);
            UpdateModemStr(connectStr);
        }
        else
        {
            UpdateModemStr(MSG("Modem Status: Connected to I
            nternet
            \n", 184));
        }
    }
}

if (DloConn == DLO_CONN_PACKAGE)
    UpdateModemStr(MSG("Modem Status: Disconnecting from Pac
    kage Delivery
    \n", 101));
else
    UpdateModemStr(MSG("Modem Status: Disconnecting from Int
    ernet
    \n", 475));

int DloGetWriteBufferSize( void )
{
    LONG writeCount = 0;

    if (!AIOWriteBufferSize)
        return(2048);

    AIOGetPortStatus(AIOPortHandle, &writeCount, NULL, NULL, NULL, NULL
    );
    return(AIOWriteBufferSize - writeCount);
}

int DloAndCommEmpty (void)
{
    if (DloConn == DLO_CONN_PACKAGE)
        return(DloPxmCount == 0);
    else
        return(DloIxmCount == 0);
}

void DloFlushReceive(void)
{
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
}

```

```

    DloRcvCount = 0;
    DloRcvIndex = 0;
    DloReadIndex = 0;

    void DloStartConn(int timeout)
    {
        if (DloNextConn == DLO_CONN_IDLE)
        {
            /* Assume package delivery connection first */
            DloNextConn = DLO_CONN_PACKAGE;

            if (timeout == DLO_PACKAGE_TIMEOUT)
            {
                DloInactivityTimer = DloCfg.pdeliv_inactivity_timer * 1
9,
            }
            else if (timeout == DLO_INET_TIMEOUT)
            {
                DloInactivityTimer = DloCfg.tinet_inactivity_timer * 19
;
                DloNextConn = DLO_CONN_INET;
            }
            else if (timeout == DLO_GETKEYS_TIMEOUT)
            {
                DloInactivityTimer = 30 * 19;
            }
            else if (timeout > 2)
            {
                DloInactivityTimer = timeout * 19;
            }
            else
            {
                DloInactivityTimer = 2 * 19;
            }
        }

        if (DloState == DLOS_CONN && DloNextConn == DloConn)
        {
            DloNextConn = DLO_CONN_IDLE;
            StateMachine(DLOE_SEND);
            return;
        }

        if (DloConn == DLO_CONN_IDLE)
        {
            StateMachine (DLOE_SEND);
            DloConn = DloNextConn;
            DloNextConn = DLO_CONN_IDLE;

            else if (DloConn == DLO_CONN_INET)
            {
                /* Package waiting for internet. Cause it to timeout quickly */
                DloStartTimer(19);
            }

            BaudRateHandler(int option, void *parameter)
            {
                parameter = parameter;
                return option;
            }

            static void NoSortHandler(LIST *head, LIST *tail, NUTInfo *handle)
            {

```

```

                head = head;
                tail = tail;
                handle = handle;

                return;
            }

            void PDConfiguration(void)
            {
                int i;
                void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
                int save;
                DloCfg_t tmpDloCfg;
                MFCNTROL *mfctl1;
                int baud;

                CMovB(&DloCfg, &tmpDloCfg, sizeof(DloCfg_t));

                NWSInitForm(NUTHandle);

                NWSGetListSortFunction(NUTHandle, &oldSortFunction);
                NWSSetListSortFunction(NUTHandle, NoSortHandler);

                i = 0;
                NWSAppendCommentField(i, 2, MSG("Package Phone
                : ", 311), NUTH
                andle);

                NWSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDloCfg.pdeliv_phone_num
                dial_chars, F_NO_HELP, NUTHandle);

                i++;
                baud = tmpDloCfg.pdeliv_baud_index;
                NWSAppendCommentField(i, 2, MSG("Package Baud
                : ", 313), NUTH
                andle);

                mfctl1 = NWSInitMenuField(InxMSG("Baud Rate", 314), 10, 40, BaudRateHand
                ler, NUTHandle);

                NWSAppendToMenuField(mfctl1, InxMSG("2400", 315), 0, NUTHandle);
                NWSAppendToMenuField(mfctl1, InxMSG("3600", 316), 1, NUTHandle);
                NWSAppendToMenuField(mfctl1, InxMSG("4800", 317), 2, NUTHandle);
                NWSAppendToMenuField(mfctl1, InxMSG("7200", 318), 3, NUTHandle);
                NWSAppendToMenuField(mfctl1, InxMSG("9600", 319), 4, NUTHandle);
                NWSAppendToMenuField(mfctl1, InxMSG("19200", 320), 5, NUTHandle);
                NWSAppendToMenuField(mfctl1, InxMSG("38400", 321), 6, NUTHandle);
                NWSAppendToMenuField(mfctl1, InxMSG("57600", 322), 7, NUTHandle);
                NWSAppendToMenuField(mfctl1, InxMSG("115200", 323), 8, NUTHandle);
                NWSAppendMenuField(i, 28, NORMAL_FIELD, &baud, mfctl1, NULL, NUTHandle);

                i++;
                NWSAppendCommentField(i, 2, MSG("Package Inactivity(sec) : ", 324), NUTH
                andle);

                NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.pdeliv_inac
                tivity_timer, 1, 65535, F_NO_HELP, NUTHandle);

                i++;
                NWSAppendCommentField(i, 2, MSG("Max Database Entries : ", 327), NUTH
                andle);

                NWSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDloCfg.max_db_entr
                ies, 1, 65535, F_NO_HELP, NUTHandle);

                i++;
                save = NWSEditPortalForm(InxMSG("Package Delivery Configuration Editor",
                308),
                12, 40,
                /* center line, column */
                i, 76,

```

```

/* form height, width */
F_VERIFY, F_NO_HELP, /* Control flags, help message */
InxMSG("Save Changes?", 328),
NUTHandle);
/* Confirm message, hand
le */

le */
NMSSetListSortFunction(NUTHandle, oldSortFunction);
NMSDestroyForm(NUTHandle);
if (!save)
    return;
tmpDlocfg.pdeliv_baud_index = baud;
ChovB(&tmpDlocfg, &Dlocfg, sizeof(Dlocfg_t));
DPCUpdateConfigFile();
)
LONG
(
    ModifyPPPConfig(FIELD *fp, int key, int *changed, NUTInfo *handle)
    int i;
    Dlocfg_t *tmpDlocfg;
    LONG save;
    key = key;
    changed = changed;
    handle = handle;
    tmpDlocfg = (Dlocfg_t *)fp->customData;
    if (NMSPushList(NUTHandle) == 0)
        return K_SELECT;
    NMSInitForm(NUTHandle);
    i = 0;
    NMSAppendCommentField(i, 2, MSG("Authentication User Name: ", 516), NUTH
andle);
    NMSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDlocfg->ppp_login,
printables, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Authentication Password: ", 578), NUTH
andle);
    NMSAppendStringField(i, 28, 30, NORMAL_FIELD, tmpDlocfg->ppp_password,
printables, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Maximum Receive Unit : ", 579), NUTH
andle);
    NMSAppendIntegerField(i, 28, NORMAL_FIELD, (int *)&tmpDlocfg->ppp_mru, 6
4, 16384, F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, MSG("Asynch. Control Char Map: 0x", 580), NU
THandle);
    NMSAppendHexField(i, 30, NORMAL_FIELD, (int *)&tmpDlocfg->ppp_accm, 0, 0
xffffffff, F_NO_HELP, NUTHandle);
    i++;
    save = NMSEditPortalForm(InxMSG("PPP Configuration Editor", 510),
12, 40,
/* center line, column */
i, 78,
/* center line & width */
F_NO_HELP, NUTHandle);
/* Confirm message, hand
le */
)
)
LONG
(
    ModifyNetConfig(FIELD *fp, int key, int *changed, NUTInfo *handle)
    int i;
    Dlocfg_t *tmpDlocfg = (Dlocfg_t *)fp->customData;
    LONG interface;
    char hw_addr[16];
    int save;
    key = key;
    changed = changed;
    handle = handle;
    /* not used */
    /* not used */
    /* not used */
    if (NMSPushList(NUTHandle) == 0)
        return K_SELECT;
    NMSInitForm(NUTHandle);
    i = 0;
    interface = tmpDlocfg->net_interface;
    NMSAppendCommentField(i, 2, "interface: ", NUTHandle);
    NMSAppendUnsignedIntegerField(i, 35, NORMAL_FIELD,
&interface,
0, 256,
F_NO_HELP, NUTHandle);
    i++;
    NMSAppendCommentField(i, 2, "Router Mac Address: ", NUTHandle);
    sprintf(hw_addr, "%02x-%02x-%02x-%02x-%02x-%02x",
tmpDlocfg->net_addr[0],
tmpDlocfg->net_addr[1],
tmpDlocfg->net_addr[2],
tmpDlocfg->net_addr[3],
tmpDlocfg->net_addr[4],
tmpDlocfg->net_addr[5]);
    NMSAppendStringField(i, 35, 17, NORMAL_FIELD,
hw_addr,
"0..9A..Fa..f-",
F_NO_HELP, NUTHandle);
    i++;
    save = NMSEditPortalForm(InxMSG("Network Route Configuration Editor", 67
9),
12, 40,
/* center line & width */
i, 78,
/* form height & width */
F_NO_HELP, NUTHandle);
/* Confirm message, hand
le */
)
)

```



```
printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait5: ", 532), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_5,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 5: ", 603), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_5, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send5: ", 534), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_5,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait6: ", 536), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_6,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 6: ", 604), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_6, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send6: ", 538), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_6,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait7: ", 540), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_7,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 7: ", 605), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_7, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send7: ", 542), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_7,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait8: ", 544), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_8,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 8: ", 606), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_8, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send8: ", 546), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_8,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Wait9: ", 548), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->wait_for_9,
    printables, F_NO_HELP, NUTHandle);
NWSAppendCommentField(i, 40, MSG("Wait Timeout 9: ", 607), NUTHandle);
NWSAppendIntegerField(i, 56, NORMAL_FIELD, (int *)&tmpDloCfg->wait_timeo
ut_9, 2, 60, F_NO_HELP, NUTHandle);
```

```
i++;
NWSAppendCommentField(i, 2, MSG("Send9: ", 550), NUTHandle);
NWSAppendStringField(i, 9, 30, NORMAL_FIELD, tmpDloCfg->send_9,
    printables, F_NO_HELP, NUTHandle);
```

```
i++;
save = NWSEditPortalForm(InxMSG("Login Script Editor", 582),
    12, 40,
    /* center line, column */
    i, 78,
    /* form height, width */
    F_NOVERIFY, F_NO_HELP, /* Control flags, help message */
    InxMSG("Save Changes?", 583),
    NUTHandle);
/* Confirm message, hand
```

```
le */
// if (save)
// {
//     CMovB(tmpDloCfg->wait_for_1, DloCfg.wait_for_1, 30*18);
//     CMovB(&tmpDloCfg->wait_timeout_1, &DloCfg.wait_timeout_1, 9 * 81
//         zeof(LONG));
//     DPUpdateConfigFile();
//     NWSDestroyForm(NUTHandle);
//     NWSPopList(NUTHandle);
//     return K_SELECT;
// }
int NewProtocolFlag = -1;
```

```
LONG ChangeProtocol(FIELD *fp, int key, int *changed, NUTInfo *handle)
{
    DloCfg_t *tmpDloCfg;
    LIST *listPtr, *slipList, *pppList, *netList;
    LONG ccode;
    LONG rcode = K_SELECT;
```

```
key = key;
changed = changed;
handle = handle;
tmpDloCfg = (DloCfg_t *)fp->customData;
if (NWSPushList(NUTHandle) == 0)
    return rcode;
NWSInitList(NUTHandle, NULL);
```

```
slipList = NWSAppendToList(MSG("Modem - SLIP", 519), (void *)OUT_SLIP, N
    UTHandle);
pppList = NWSAppendToList(MSG("Modem - PPP", 521), (void *)OUT_PPP, NUTH
    andle);
netList = NWSAppendToList(MSG("LAN/WAN", 537), (void *)OUT_NETWORK, NUTH
    andle);
```

```
if (tmpDloCfg->out_protocol == OUT_SLIP)
{
    listPtr = slipList;
}
else if (tmpDloCfg->out_protocol == OUT_PPP)
{
    listPtr = pppList;
}
else
{
    listPtr = netList;
}
ccode = NWSList(
```

```

InxMSG("Outbound Protocol", 509),
12, 40,
3,
16,
M_ESCAPE | M_SELECT,
&listPtr,
NUTHandle, NULL,
NULL, NULL);

if (ccode == M_SELECT)
{
    tmpDioCfg->out_protocol = NewProtocolFlag = (int)listPtr->other1;
    rcode = K_ESCAPE;
}

NWSDestroyList(NUTHandle);
NWSPopList(NUTHandle);
return rcode;
}

//LONG ChangeTunnel(FIELD *fp, int key, int *changed, NUTInfo *handle)
//{
//    DioCfg_t *tmpDioCfg;
//    LIST *listPtr, *enableList, *disableList;
//    LONG ccode;
//    LONG rcode = K_SELECT;
//
//    key = key;
//    changed = changed;
//    handle = handle;
//
//    tmpDioCfg = (DioCfg_t *)fp->customData;
//
//    if (NWSPushList(NUTHandle) == 0)
//        return rcode;
//
//    NWSInitList(NUTHandle, NULL);
//
//    enableList = NWSAppendToList(MSG("Enabled", 624), (void *)1, NUTHandle);
//    disableList = NWSAppendToList(MSG("Disabled", 625), (void *)0, NUTHandle);
//
//    if (tmpDioCfg->tunnel)
//    {
//        listPtr = enableList;
//    }
//    else
//    {
//        listPtr = disableList;
//    }
//
//    ccode = NWSList(
//        InxMSG("Tunnel Header", 626),
//        12, 40,
//        2,
//        16,
//        M_ESCAPE | M_SELECT,
//        &listPtr,
//        NUTHandle, NULL,
//        NULL, NULL);
//
//    if (ccode == M_SELECT)
//    {
//        tmpDioCfg->tunnel = NewProtocolFlag = (int)listPtr->otherInfo;
//        rcode = K_ESCAPE;
//    }
//
//    InxMSG("Outbound Protocol", 509),
//    12, 40,
//    3,
//    16,
//    M_ESCAPE | M_SELECT,
//    &listPtr,
//    NUTHandle, NULL,
//    NULL, NULL);
//
//    if (ccode == M_SELECT)
//    {
//        tmpDioCfg->out_protocol = NewProtocolFlag = (int)listPtr->other1;
//        rcode = K_ESCAPE;
//    }
//
//    NWSDestroyList(NUTHandle);
//    NWSPopList(NUTHandle);
//    return rcode;
//}

void ProviderConfiguration(void)
{
    int i;
    void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);
    int save; /*
    DioCfg_t tmpDioCfg;
    int ip0, ip1, ip2, ip3;
    int gw0, gw1, gw2, gw3;
    MFCNTROL *mfct10;
    int baud;
    FIELD *fp;
    char *protocolStr;
    char *pppConfigStr;
    int cflags = F_VERIFY;
    char *tunnelStr;

    CMovB(&DioCfg, &tmpDioCfg, sizeof(DioCfg_t));

    NewProtocolLoop:
    NewProtocolFlag = -1;
    NWSInitForm(NUTHandle);
    NWSGetListSortFunction(NUTHandle, &oldSortFunction);
    NWSSetListSortFunction(NUTHandle, NoSortHandler);

    i = 0;
    NWSAppendCommentField(i, 30, MSG("Outbound Protocol", 525), NUTHandle);

    if (tmpDioCfg.out_protocol == OUT_SLIP)
        protocolStr = MSG("Modem - SLIP", 527);
    else if (tmpDioCfg.out_protocol == OUT_PPP)
        protocolStr = MSG("Modem - PPP", 529);
    else
        protocolStr = MSG("LAN/WAN", 584);

    fp = NWSAppendHotSpotField(i, 50, NORMAL_FIELD, protocolStr,
        ChangeProtocol, NUTHandle);
    fp->customData = &tmpDioCfg;

    i++;
    NWSAppendCommentField(i, 2, MSG("Internet Phone", 126), NUTHandle);
    NWSAppendStringField(i, 30, 30, NORMAL_FIELD, tmpDioCfg.tinet_phone_num,
        dial_chars, F_NO_HELP, NUTHandle);

    i++;
    baud = tmpDioCfg.tinet_baud_index;
    NWSAppendCommentField(i, 2, MSG("Internet Baud", 128), NUTHandle);
    mfct10 = NWSInitMenuField(InxMSG("Baud Rate", 129), 10, 40, BaudRateHandler, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("2400", 130), 0, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("3600", 131), 1, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("4800", 132), 2, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("7200", 133), 3, NUTHandle);
    NWSAppendToMenuField(mfct10, InxMSG("9600", 146), 4, NUTHandle);
}

```

```

NWSAppendToMenuField(mfct10, InxMSG("19200", 297), 5, NUTHANDLE);
NWSAppendToMenuField(mfct10, InxMSG("38400", 300), 6, NUTHANDLE);
NWSAppendToMenuField(mfct10, InxMSG("57600", 304), 7, NUTHANDLE);
NWSAppendToMenuField(mfct10, InxMSG("115200", 305), 8, NUTHANDLE);
NWSAppendMenuField(i, 30, NORMAL_FIELD, &baud, mfct10, NULL, NUTHANDLE);

i++;
NWSAppendCommentField(i, 2, MSG("Internet MTU : ", 309), NU
THANDLE);
NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&tmpDlOCfg.mtu, 1, 655
35, F_NO_HELP, NUTHANDLE);

i++;
NWSAppendCommentField(i, 2, MSG("Internet Inactivity(sec) : ", 310), NU
THANDLE);
NWSAppendIntegerField(i, 30, NORMAL_FIELD, (int *)&tmpDlOCfg.tinet_inact
ivity_timer, 1, 65535, F_NO_HELP, NUTHANDLE);

//
// NWSAppendCommentField(i, 2, MSG("IP to IP tunneling : ", 537), NU
THANDLE);
// NWSAppendBoolField(i, 30, NORMAL_FIELD, (BYTE *)&tmpDlOCfg.tunnel, F_NO_
HELP, NUTHANDLE);
// if (tmpDlOCfg.tunnel)
//     tunnelStr = MSG("Enabled", 627);
// else
//     tunnelStr = MSG("Disabled", 628);
//
// fp = NWSAppendHotSpotField(i, 30, NORMAL_FIELD, tunnelStr,
// ChangeTunnel, NUTHANDLE);
// fp->customData = &tmpDlOCfg;
// if (tmpDlOCfg.tunnel)
// {
//     LONG ip_address = ntohl(tmpDlOCfg.ip_address);
//     LONG gateway_address = ntohl(tmpDlOCfg.gateway_address);
//     i++;
//     ip0 = (ip_address >> 8) & 0xff;
//     ip1 = (ip_address >> 16) & 0xff;
//     ip2 = (ip_address >> 24) & 0xff;
//     ip3 = (ip_address >> 32) & 0xff;
//     NWSAppendCommentField(i, 2, MSG("IP Address(ISP) :
//     ", 306), NUTHANDLE);
//     NWSAppendIntegerField(i, 30, NORMAL_FIELD, &ip3, 1, 255, F_NO_HE
LP, NUTHANDLE);
//     NWSAppendIntegerField(i, 34, NORMAL_FIELD, &ip2, 1, 255, F_NO_HE
LP, NUTHANDLE);
//     NWSAppendIntegerField(i, 38, NORMAL_FIELD, &ip1, 1, 255, F_NO_HE
LP, NUTHANDLE);
//     NWSAppendIntegerField(i, 42, NORMAL_FIELD, &ip0, 1, 255, F_NO_HE
LP, NUTHANDLE);
//
//     i++;
//     gw0 = (gateway_address >> 8) & 0xff;
//     gw1 = (gateway_address >> 16) & 0xff;
//     gw2 = (gateway_address >> 24) & 0xff;
//     gw3 = (gateway_address >> 32) & 0xff;
//     NWSAppendCommentField(i, 2, MSG("Hybrid Gateway Address :
//     ", 307), NUTHANDLE);
//     NWSAppendIntegerField(i, 30, NORMAL_FIELD, &gw3, 1, 255, F_NO_HE
LP, NUTHANDLE);
//     NWSAppendIntegerField(i, 34, NORMAL_FIELD, &gw2, 1, 255, F_NO_HE
LP, NUTHANDLE);
//     NWSAppendIntegerField(i, 38, NORMAL_FIELD, &gw1, 1, 255, F_NO_HE
LP, NUTHANDLE);
//     NWSAppendIntegerField(i, 42, NORMAL_FIELD, &gw0, 1, 255, F_NO_HE
LP, NUTHANDLE);
}

InxMSG("Save Changes?", 556)*/ NULL, /* Confirm message, hand
NUTHANDLE);

/* save */ NWSeditPortalForm(InxMSG("Provider Configuration Editor", 55
12, 40,
/* center line, column */
1, 78,
/* form height, width */
/* cflags*/F_NOVERIFY, F_NO_HELP,
ol flags, help message */
/* InxMSG("Save Changes?", 556)*/ NULL, /* Confirm message, hand
NUTHANDLE);

NWSSetListSortFunction(NUTHANDLE, oldSortFunction);
NWSDestroyForm(NUTHANDLE);

if (NewProtocolFlag != -1)
{
    cflags = F_FORCE;
    goto NewProtocolLoop;
}

if (!save)
    return;

tmpDlOCfg.ip_address = htonl((ip0) |
(ip1 << 8) |
(ip2 << 16) |

```

```

tmpDioCfg.gateway_address = htonl((gw0) |
    (gw1 << 8) |
    (gw2 << 16) |
    (gw3 << 24));

tmpDioCfg.tinet_baud_index = baud;

if (memcmp(&DioCfg, &tmpDioCfg, sizeof(DioCfg)) == 0 ||
    NWSConfirm(InxMSG("Save Changes?", 612), 0, 0, TRUE, NULL,
        NUTHandle, NULL) == FALSE)
    return;

/*
 * Get newest wait_for and send strings in case ModifyLoginsScript()
 * changed them.
 */

// CMovB(DioCfg.wait_for_1, tmpDioCfg.wait_for_1, 30 * 18);
// CMovB(DioCfg.wait_timeout_1, &tmpDioCfg.wait_timeout_1, 9 * sizeof(LONG));
// );

CMovB(&tmpDioCfg, &DioCfg, sizeof(DioCfg.t));

InetChangeProtocol();

DPCUpdateConfigFile();

void ModemConfiguration(void)
{
    int i;
    int save;
    DioCfg_t tmpDioCfg;
    void (*oldSortFunction)(LIST *, LIST *, NUTInfo *);

    CMovB(&DioCfg, &tmpDioCfg, sizeof(DioCfg.t));

    NWSInitForm(NUTHandle);

    NWSGetListSortFunction(NUTHandle, &oldSortFunction);
    NWSSetListSortFunction(NUTHandle, NoSortHandler);

    i = 0;
    NWSAppendCommentField(i, 2, MSG("Packet Life(sec) : ", 325), NUTHandle);

    NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDioCfg.packet_life_time, 1, 65535, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Call Setup(sec) : ", 326), NUTHandle);

    NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDioCfg.call_setup_timeout, 1, 65535, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Async Buffer Size : ", 212), NUTHandle);

    NWSAppendIntegerField(i, 24, NORMAL_FIELD, (int *)&tmpDioCfg.async_buffer_size, 1500, 1024*100, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Dial Prefix : ", 329), NUTHandle);

    NWSAppendStringField(i, 24, 10, NORMAL_FIELD, tmpDioCfg.dialout_prefix,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Hangup Str : ", 330), NUTHandle);

    NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDioCfg.hangup_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Disconnect str : ", 331), NUTHandle);

    NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDioCfg.disconnect_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Escape Str : ", 332), NUTHandle);

    NWSAppendStringField(i, 24, 20, NORMAL_FIELD, tmpDioCfg.escape_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Connect str : ", 333), NUTHandle);

    NWSAppendStringField(i, 24, 50, NORMAL_FIELD, tmpDioCfg.connect_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    NWSAppendCommentField(i, 2, MSG("Init str : ", 334), NUTHandle);

    NWSAppendStringField(i, 24, 60, NORMAL_FIELD, tmpDioCfg.init_str,
        modem_control_chars, F_NO_HELP, NUTHandle);

    i++;
    save = NWSeditPortalForm(InxMSG("Modem Configuration Editor", 513),
        12, 40,
        /* center line, column */
        i, 76,
        /* form height, width */
        F_VERIFY, F_NO_HELP,
        /* Control flags, help message */
        InxMSG("Save Changes?", 514),
        NUTHandle);

    /* Confirm message */

    NWSSetListSortFunction(NUTHandle, oldSortFunction);
    NWSDestroyForm(NUTHandle);

    if (!save)
        return;

    CMovB(&tmpDioCfg, &DioCfg, sizeof(DioCfg.t));

    if (AIOPortHandle != -1)
    {
        AIOWriteBufferSize(AIOPortHandle, DioCfg.async_buffer_size);
        AIOWriteBufferSize(AIOPortHandle, &AIOWriteBufferSize);
        DioMaxBufferSize = (AIOWriteBufferSize < DIOBUFSIZE) ? AIOWriteBufferSize : DIOBUFSIZE;
        DioMaxBufferSize = (AIOWriteBufferSize < DIOBUFSIZE) ? AIOWriteBufferSize : DIOBUFSIZE;
    }

    DPCUpdateConfigFile();
}

```

```

void DPCCConfiguration(void)
{
    LIST *listPtr = NULL;
    LONG ccode = M_SELECT;
    NWSInitList(NUTHandle, NULL);
    NWSAppendToList(MSG("Modem Configuration", 263), (void *)1, NUTHandle);
    NWSAppendToList(MSG("Package Delivery Configuration", 507), (void *)2, NUTHandle);
    NWSAppendToList(MSG("Provider Configuration", 508), (void *)3, NUTHandle);
    while (ccode != M_ESCAPE)
    {
        ccode = NWSList(
            InxMSG("DPC Configuration", 533),
            12, 40,
            3,
            32,
            M_ESCAPE | M_SELECT,
            &listPtr,
            NUTHandle, NULL,
            NULL, NULL);

        if (ccode == M_SELECT)
        {
            if (NWSPushList(NUTHandle) != 0)
            {
                switch ((int)listPtr->otherInfo)
                {
                    case 1: ModemConfiguration();
                           break;

                    case 2: PDCConfiguration();
                           break;

                    case 3: ProviderConfiguration();
                           break;

                    default: break;
                }
                NWSPopList(NUTHandle);
            }
            NWSDestroyList(NUTHandle);
        }
    }
}

/*****
 *
 * DioReceive(BYTE *buffer
 *           int size)
 *
 * Description:
 *   This routine reads a bytes from the global modem receive buffer
 *   and returns it to the caller.
 *
 * Input:
 *
 *   - max number of bytes to
 *   - buffer to write data into
 *
 * Returns:
 *   Number of bytes read
 *
 * *****/
int DioGetRcvBytesBuffered(void)
{
    return(DioRcvCount);
}

int DioReceive (BYTE *buffer, int size)
{
    int i, iRetSize = 0;
    if(DioRcvCount)
    {
        iRetSize = (DioRcvCount > size)? size : DioRcvCount;
        DioRcvCount -= iRetSize;
        for(i=0; i<iRetSize; i++)
        {
            buffer[i] = DioRcvBuffer[DioReadIndex++];
            if(DioReadIndex >= DIOBUFSIZE)
                DioReadIndex = 0;
        }
    }

    return iRetSize;
}

/*****
 *
 * DioEmpty(void)
 *
 * Description:
 *   This routine returns true if the modem transmit buffer is empty,
 *   meaning that all previous requests have been sent.
 *
 * Input:
 *   nothing
 *
 * Output:
 *   nothing
 *
 * Returns:
 *   TRUE if transmit buffer is empty
 *
 * *****/
int DioEmpty (void)
{
    if (DioConn == DLO_CONN_PACKAGE)
        return (DioPxmCount == 0);
    else
        return (DioIxmCount == 0);
}

/*****
 *
 * WriteCommPhoneNumber(void)
 *
 * *****/

```

```

* Description:
* This routine is called when the modem needs to be dailed.

```

```

* Input:

```

```

* nothing

```

```

* Output:

```

```

* nothing

```

```

* Returns:

```

```

* nothing

```

```

* .....

```

```

static void WriteCommPhoneNumber(void)

```

```

{
    int baudIndex, i;
    char *number;

```

```

    if (DloConn == DLO_CONN_PACKAGE)

```

```

        baudIndex = DloCfg.pdeliv_baud_index;

```

```

    else
        baudIndex = DloCfg.tinet_baud_index;

```

```

    AIOConfigurePort(AIOPortHandle,
        AIOBaudRateDefines[baudIndex],
        AIO_DATA_BITS_8, AIO_STOP_BITS_1,
        AIO_PARITY_NONE,
        AIO_SOFTWARE_FLOW_CONTROL_OFF | AIO_HARDWARE_FLOW_CONTROL_ON);

```

```

    if (DloCfg.dialout_prefix[0])

```

```

    {
        if (DloConn == DLO_CONN_PACKAGE)
            number = DloCfg.pdeliv_phone_num;

```

```

        else
            number = DloCfg.tinet_phone_num;

```

```

        for (i = 0; number[i]; i++)
        {

```

```

            if (number[i] < 'A' || number[i] > 'z')
                break;

```

```

            if (number[i] > 'z' && number[i] < 'a')
                break;

```

```

        }

```

```

        if (i)

```

```

            SendAIOData(number, i); /* Send the alpha string

```

```

    );
    SendAIOData(DloCfg.dialout_prefix, CStrLen(DloCfg.dialout_prefix));
    SendAIOData(&number[i], CStrLen(&number[i]));

```

```

}

```

```

else

```

```

{
    if (DloConn == DLO_CONN_PACKAGE)
        SendAIOData(DloCfg.pdeliv_phone_num, CStrLen(DloCfg.pdeliv_phone_num));

```

```

    else
        SendAIOData(DloCfg.tinet_phone_num, CStrLen(DloCfg.tinet_phone_num));

```

```

}

```

```

    SendAIOData(MSG("\r", 613), 1);

```

```

    if (DloState == DLOS_REDL)

```

```

    {

```

```

    }

```

```

    }

```

```

    }

```

```

UpdateModemStr(MSG("Modem Status: Redialing

```

```

\n", 559));

```

```

else

```

```

    UpdateModemStr(MSG("Modem Status: Dialing

```

```

\n", 560));

```

```

}

```

```

/* .....

```

```

* .....

```

```

* ProcessDisconnect(void)

```

```

* .....

```

```

* Description:

```

```

* This routine is called when where attaching and we lose Carrier

```

```

* Detect.

```

```

* .....

```

```

* Input:

```

```

* nothing

```

```

* .....

```

```

* Output:

```

```

* nothing

```

```

* .....

```

```

* Returns:

```

```

* nothing

```

```

* .....

```

```

static void ProcessDisconnect(void)

```

```

{
    int count;

```

```

    if (DloConn == DLO_CONN_PACKAGE)

```

```

    {

```

```

        UpdateModemStr(MSG("Modem Status: Disconnected from Package Deli

```

```

\n", 237));

```

```

        count = DioPxmItCount;

```

```

    }

```

```

    else

```

```

    {

```

```

        UpdateModemStr(MSG("Modem Status: Disconnected from Internet

```

```

\n", 472));

```

```

        count = DioIXmitCount;

```

```

    }

```

```

    if (count)

```

```

    {

```

```

        WriteCommPhoneNumber();

```

```

        DioStartTimer(DloCfg.call_setup_timeout * 19);

```

```

        DloState = DLOS_DIAL;

```

```

        if (DloConn == DLO_CONN_INET)

```

```

            InetStateChange(DLOS_DIAL);

```

```

        }

```

```

    else

```

```

    {

```

```

        DioStartTimer(1 * 19);

```

```

        DloState = DLOS_DISC_4;

```

```

        if (DloConn == DLO_CONN_INET)

```

```

            InetStateChange(DLOS_DISC_4);

```

```

        }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

```

    }

```

This routine is terminate a modem connection.

Input: nothing
Output: nothing
Returns: nothing

void DloEndConn(void)

```
{
    if (AIOPortHandle < 0)
        return;
    if (DloConn == DLO_CONN_PACKAGE)
        DloPxmItCount = 0;
    else
        DloIXmitCount = 0;
```

switch(DloState)

```
{
    case DLOS_INIT:
    case DLOS_REDL:
    case DLOS_DIAL:
    case DLOS_CONN:
```

AIOFlushBuffers(AIOPortHandle, (AIO_FLUSH_WRITE_BUFFER |

```
AIO_FLUSH_READ_BUFFER));
    DloState = DLOS_CONN;
    StateMachine(DLOE_TIMEOUT);
    break;
```

```
case DLOS_IDLE:
case DLOS_DISC_1:
case DLOS_DISC_2:
case DLOS_DISC_3:
case DLOS_DISC_4:
    StateMachine(DLOE_DISCONN);
    break;
}
```

_0003(void) In IDLE state, got SND event

Description:
The state machine is in the IDLE state.
We've received a SND request.

Input: nothing
Output: nothing
Returns: nothing

int DloConnected(void)

```
{
    ed to Internet at%.24s\n", 563), ConnectBaudStr);
```

LONG extStatus = 0, chgdExtStatus;

```
if (AIOPortHandle < 0)
    return(FALSE);
```

```
AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus);
if (extStatus & AIO_EXTSTA_DCD)
    return(TRUE);
```

```
return(FALSE);
}
```

static void _0003 (void)

```
{
    LONG extStatus, chgdExtStatus;
```

```
InitializeAIO();
if (AIOPortHandle < 0)
```

```
{
    UpdateModemStr(MSG("Modem Status: ERROR - Unable to initialize A
\n", 238));
    return;
}
```

```
if (AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus)
    || !(extStatus & AIO_EXTSTA_DCD))
```

```
{
    AIOFlushBuffers(AIOPortHandle,
        (AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER)
    );
```

```
SendAIOData(DloCfg.Init_str, CStrlen(DloCfg.Init_str));
SendAIOData(MSG("\r", 614), 1);
DloStartTimer(5 * 19);
```

```
DloState = DLOS_INIT;
```

```
if (DloConn == DLO_CONN_INET)
```

```
InetStateChange(DLOS_INIT);
```

```
UpdateModemStr(MSG("Modem Status: Initializing Modem
\n", 561));
}
```

```
else
```

```
{
    if (DloConn == DLO_CONN_PACKAGE)
```

```
DloStartTimer(DloInactivityTimer);
```

```
else
```

```
DloStartTimer(DloInactivityTimer);
```

```
DloStartTimer(30*19);
```

```
DloStopPacketLifeTimer();
```

```
WriteCommXmitBuffer();
```

```
DloState = DLOS_CONN;
```

```
if (DloConn == DLO_CONN_INET)
```

```
InetStateChange(DLOS_CONN);
```

```
if (DloConn == DLO_CONN_PACKAGE)
```

```
UpdateModemStr(MSG("Modem Status: Connected to Package D
\n", 562));
```

```
elivary
```

```
else
```

```
UpdateModemStr(MSG("Modem Status: Connected to Internet
\n", 473));
```

```
{
    if (ConnectBaudStr[0])
```

```
{
    BYTE connectStr[80];
```

```
NWSprintf(connectStr, MSG("Modem Status: Connect
ed to Internet at%.24s\n", 563), ConnectBaudStr);
```

UpdateModemStr(connectStr);

```
)
else
{
    UpdateModemStr(MSG("Modem Status: Connected to I
    \n", 564));
}
```

nternet

```
)
)
/.....
* _0100(void) In INIT state, got TIMEOUT
*
* Description:
* The state machine is in the INIT state.
* We timed out waiting for the modem init sequence.
*
* Input: nothing
*
* Output: nothing
*
* Returns: nothing
*
* .....
```

In INIT state, got TIMEOUT

Description:
The state machine is in the INIT state.
We timed out waiting for the modem init sequence.

Input: nothing
Output: nothing
Returns: nothing

```
...../
```

static void _0100(void)

```
{
    LONG extStatus, chgdExtStatus;
```

```
if (AIOGetExternalStatus( AIOPortHandle, &extStatus, &chgdExtStatus)
    || !(extStatus & AIO_EXTSTA_DCD))
```

```
{
    WriteCommPhoneNumber();
    DloStartTimer(Dlocfg.call_setup_timeout * 19);
    DioState = DLOS_DIAL;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_DIAL);
}
```

}

else

```
{
    UpdateModemStr(MSG("Modem Status: Error - Still Connected
    \n", 242));
    DioEndConn();
}
```

```
...../
```

```
* _0104(void) In INIT state, got OK response from mode
```

Description:
The state machine is in the INIT state.
We've received an OK response from sending modem init sequence.

Input: nothing
Output: nothing

```
...../
```

```
.....
* Returns: nothing
*
* .....
```

static void _0104(void)

```
{
    WriteCommPhoneNumber();
```

```
    DloStartTimer(Dlocfg.call_setup_timeout * 19);
    DioState = DLOS_DIAL;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_DIAL);
}
```

```
...../
```

```
* _0200(void) In DIAL state, got TIMEOUT
```

Description:
The state machine is in the DIAL state.
It timed out waiting for connect.

Input: nothing
Output: nothing
Returns: nothing

```
...../
```

static void _0200(void)

```
{
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_WRITE_BUFFER);
    AIOFlushBuffers(AIOPortHandle, AIO_FLUSH_READ_BUFFER);
    SendAIOData(MSG("\r", 615), 1);
    DloStartTimer(10 * 18);
    DioState = DLOS_REDL;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_REDL);
}
```

```
...../
```

```
* _0201(void)
```

```
modem In DIAL state, got CONNECT response from
```

```
.....
* Description:
```

The state machine is in the DIAL state.
We've received a CONNECT response from sending ATDT sequence.

Input: nothing
Output: nothing
Returns: nothing

static void _0201(void)


```

* nothing
*
* Returns: nothing
*
* ...../
static void _0400(void)
(
    DloStartTimer( (1 * 19) + 9); /* 1.5 second delay */
    DloState = DLOS_DISC_1;
    if (DloNextConn == DloConn)
        DloNextConn = DLO_CONN_IDLE;
    if (DloConn == DLO_CONN_INET) {
        UpdateModemStr(MSG("Modem Status: Disconnecting from Internet
        \n", 566));
        InetStateChange(DLOS_DISC_1);
    }
    else if (DloConn == DLO_CONN_PACKAGE)
        UpdateModemStr(MSG("Modem Status: Disconnecting from Package Delivery
        \n", 565));
    else
        UpdateModemStr("Modem Status: Disconnecting
        \n");
)
/...../
*
* _0402(void) In CONNECT state, got disconnected
*
* Description:
* State: The state machine is in the CONNECT state.
* Event: We were disconnected by the remote site.
* Action: If still have data to send:
*         Send connect string(ATDT
1800...), DLOS_DIAL state.
*         else
*         Start 1 second timer, DL
OS_DISC_4 state.
*
* Input: nothing
*
* Output: nothing
*
* Returns: nothing
*
* ...../
static void _0402(void)
(
    ProcessDisconnect();
)
/...../
*
* _0403(void) In CONNECTED state, got SEND request
*
* Description:
* State: The state machine is in the CONNECTED state.
* Event: We've received a request to send data to the remote site
* Action: Extend the inactivity timer.

```

```

* Input: nothing
*
* Output: nothing
*
* Returns: nothing
*
* ...../
static void _0403(void)
(
    if (DloConn == DLO_CONN_PACKAGE)
        DloStartTimer(DloInactivityTimer);
    else
        DloStartTimer(DloInactivityTimer);
)
/...../
*
* _0500(void) In Disconnect 1 state, got timed out
*
* Description:
* State: Disconnect 1 state
*         Two ways to get in:
*         1)
*             - Inactivity timer kicke
*             - Set 1.5 pre-escape tim
*         2)
*             - Inactivity timer kicked in
*             - Set 1.5 pre-escape timer(DLO
*               second timer(DLOS_DISC_2)
*               0 second timer(DLOS_DISC_3)
*               Event: Intentional 1.5 or 10 second timeout.
*               Action: Send escape string(+++) set 2 second timer, DLOS
state.
*
* Input: nothing
*
* Output: nothing
*
* Returns: nothing
*
* ...../
static void _0500(void)
(
    AIOFlushBuffers(AIOPortHandle,
        (AIO_FLUSH_WRITE_BUFFER | AIO_FLUSH_READ_BUFFER));
    if (DebugFlag)
        fputs(DloCfg.escape_str, stdout);
    SendAIOData(DloCfg.escape_str, CStrlen(DloCfg.escape_str));
    DloStartTimer(2 * 19);
}

```

```

    DloState = DLOS_DISC_2;
    if (DloConn == DLO_CONN_INET)
        InetStateChange(DLOS_DISC_2);
}

/*.....*/
*
*   _0502(void)           In Disconnect 1 state, got disconnected
*
*   Description:
*       State: Disconnect 1 state.
*
*       Two ways to get in:
*       1)
*           - Inactivity timer kicked
*           - Set 1.5 pre-escape timer
*
*       2)
*           - Inactivity timer kicked in
*           - Set 1.5 pre-escape timer(DLO
*
*   S_DISC_1)
*
*   second timer(DLOS_DISC_2)
*
*   0 second timer(DLOS_DISC_3)
*
*   Event: We were disconnected by the remote site.
*   Action: If still have data to send:
*
*       Send connect string(ATDT
*
*       else
*           Start 1 second timer, DL
*
*   OS_DISC_4 state.
*
*   Input:   nothing
*
*   Output:  nothing
*
*   Returns: nothing
*
*   .....*/
static void _0502(void)
{
    ProcessDisconnect();
}

/*.....*/
*
*   _0600(void)           In Disconnect 2 state, timed out
*
*   Description:
*       State: Disconnect 2 state:
*
*           - Inactivity timer kicked in
*           - Set 1.5 pre-escape timer(DLO
*
*   S_DISC_1)
*
*   second timer(DLOS_DISC_2)
*
*   Event: We timed out, modem didn't respond.
*   Action: Send Hangup string with 10 second timer, DLOS_DISC_3 sta
te.
*
*   .....*/
static void _0600(void)
{
    In Disconnect 2 state, timed out
}

/*.....*/
*
*   _0602(void)           In Disconnect 2 state, got disconnected
*
*   Description:
*       State: Disconnect 2 state:
*
*           - Inactivity timer kicked in
*           - Set 1.5 pre-escape timer(DLO
*
*   S_DISC_1)
*
*   second timer(DLOS_DISC_2)
*
*   Event: We got disconnected by the remote site while waiting aft
er
*
*   Action: If still have data to send:
*           sending +++.
*           Send connect string(ATDT
*
*           else
*           Start 1 second timer, DL
*
*   OS_DISC_4 state.
*
*   Input:   nothing
*
*   Output:  nothing
*
*   Returns: nothing
*
*   .....*/
static void _0602(void)
{
    ProcessDisconnect();
}

/*.....*/
*
*   _0604(void)           In Disconnect 2 state, got OK response f
rom modem
*
*   Description:
*       State: Disconnect 2 state:

```



```

BYTE *xmitBuffer;

if (DloNextConn == DLO_CONN_IDLE)
{
    /* Assume package delivery connection first */
    DloNextConn = DLO_CONN_PACKAGE;

    if (timeout == DLO_PACKAGE_TIMEOUT)
    {
        DloInactivityTimer = DloCfg.pdeliv_inactivity_timer * 1
    }
    else if (timeout == DLO_INET_TIMEOUT ||
             timeout == DLO_INET_NO_TIMEOUT)
    {
        DloInactivityTimer = DloCfg.tinet_inactivity_timer * 19
    }
    DloNextConn = DLO_CONN_INET;
    else if (timeout == DLO_GETKEYS_TIMEOUT)
    {
        DloInactivityTimer = 30 * 19;
    }
    else if (timeout > 2)
    {
        DloInactivityTimer = timeout * 19;
    }
    else
    {
        DloInactivityTimer = 2 * 19;
    }
}

/* USE_AIO_DEADMAN
*/
* Update AIO deadman timer to timeout + 5 seconds
*/

AIOSetExternalControl(AIOPortHandle, AIO_SET_DEADMAN_TIMER, timeout + 5)

#endif

if (size > DLOBUFSIZE || size < 1)
    return 0;

if (DloState == DLOS_CONN && DloNextConn == DloConn)
{
    if (timeout != DLO_INET_NO_TIMEOUT)
    {
        DloNextConn = DLO_CONN_IDLE;
        StateMachine(DLOE_SEND);
    }
    UpdateModemLights(1, 0, 1);
    if (DebugFlag)
    {
        printf(MSG("Sending %d bytes:\n", 485), size);
        HexAsciiDump(buffer,
                      (DloConn == DLO_CONN_INET && size > 32) ? 3
                      : size);
    }
    SendAIOData(buffer, size);
    return size;
}

if (DloNextConn == DLO_CONN_PACKAGE)
{
    maxBufferSize = &DloMaxBufferSize;
    xmitCount = &DloPxmmitCount;
    xmitBuffer = DloPxmmitBuffer;
}
else
{
    maxBufferSize = &DloMaxBufferSize;
    xmitCount = &DloIXmitCount;
    xmitBuffer = DloIXmitBuffer;
}

if (size >= *maxBufferSize - *xmitCount)
    return 0;
for (i = 0; i < size) && (*xmitCount < *maxBufferSize); i++, (*xmitCount
t)++)
{
    if (!*xmitCount)
        DloStartPacketLifeTimer();
    xmitBuffer[*xmitCount] = buffer[i];
}

if (DloConn == DLO_CONN_IDLE)
{
    StateMachine (DLOE_SEND);
    DloConn = DloNextConn;
    DloNextConn = DLO_CONN_IDLE;
}
else if (DloConn == DLO_CONN_INET)
{
    /* Package waiting for internet. Cause it to timeout quickly */
    DloStartTimer(19);
}
return size;
}

/*.....
* WriteCommXmitBuffer(void)
*
* Description:
*   This routine is called after the modem is connected to send the
*   data stored in DloPxmmitBuffer to the modem.
*
* Input:
*   nothing
*
* Output:
*   nothing
*
* Returns:
*   nothing
*
*.....
static void WriteCommXmitBuffer( void )
{
    BYTE *xmitBuffer;
    LONG *xmitCount;

    if (DloConn == DLO_CONN_PACKAGE)
    {
        xmitBuffer = DloPxmmitBuffer;
        xmitCount = &DloPxmmitCount;
    }
}

```

```

)
else
{
    xmitBuffer = DloXmitBuffer;
    xmitCount = &DloXmitCount;
}

UpdateModemLights(1, 0, 1);

if (*xmitCount)
{
    if (DebugFlag)
    {
        printf(MSG("xmit %d bytes:\n", 486), *xmitCount);
        HexAsciiDump(xmitBuffer,
            (DloConn == DLO_CONN_INET && *xmitCount > 3)
        );
    }

    SendAIOData(xmitBuffer, *xmitCount);
    *xmitCount = 0;
}

/*****
 *
 * DloStartPacketLifeTimer(void)
 *
 * Description:
 *   This routine starts the Packet Life timer.
 *   This timer is set to the configured packet timer(120 seconds by
 *   default) each time data is added via DioSend.
 *   DloMain will periodically decrement the timer and clear the
 *   data buffer when it reaches zero.
 *
 * Input:      nothing
 *
 * Output:     nothing
 *
 * Returns:    nothing
 *
 *****/

static void DloStartPacketLifeTimer( void )
{
    DloPacketLifeTimer = (DloCfg.packet_lifetime * 19 > MIN_TIMER_VALUE ?
        DloCfg.packet_lifetime * 19 : MIN_TIMER_VALUE);
}

/*****
 *
 * DloStopPacketLifeTimer(void)
 *
 * Description:
 *   The routine is called to stop the packet life timer.
 *
 * Input:      nothing
 *
 * Output:     nothing
 *
 * Returns:    nothing
 *
 *****/

static void DloStopPacketLifeTimer( void )
{
    DloPacketLifeTimer = 0;
}

/*****
 *
 * DioMain
 *
 * Description:
 *   This routine is called to set the value of the state machine tim
 *   DioMain periodically decrements this timer and set the event
 *   event when it hits zero.
 *
 * Input:      ticks
 *
 * Output:     nothing
 *
 * Returns:    nothing
 *
 *****/

static void DioMain( LONG ticks )
{
    DioTimer = (ticks > MIN_TIMER_VALUE ? ticks : MIN_TIMER_VALUE);
    #if TRACE_STATE
    {
        char traceStr[20];
        NWSprintf(traceStr, MSG("%d", 611), DioTimer);
        fputs(traceStr, stdout);
    }
    #endif
}

/*****
 *
 * DioStopTimer(void)
 *
 * Input:      DioTimer = 0;
 *
 *****/

/*****
 *
 * SendAIOData(BYTE *data,
 *             int length)
 *
 * Description:
 *   The routine sends the data pointed to by data to the modem.
 *
 * Input:      data
 *             length
 *
 * Output:     nothing
 *
 * Returns:    nothing
 *
 *****/

- data to send
- size of data

```

```
void SendAIOData(BYTE *data, LONG length)
```

```
{
    LONG count;
    int bufferSize;
    BYTE *ptr;
    WORD state;
    LONG bytesWritten;
```

```
    ptr = data;
    while (ptr < (data + length))
```

```
    {
        AIOWriteStatus(AIOPortHandle, &count, &state);
        bufferSize = AIOWriteBufferSize - count;
```

```
        if (length < bufferSize)
            bufferSize = length;
```

```
        if (bufferSize > 0)
```

```
        {
            AIOWriteData(AIOPortHandle, ptr, bufferSize, &bytesWritten);
```

```
            ptr += bufferSize;
```

```
        }
        else
```

```
            ThreadSwitchWithDelay();
```

```
    }
```

```
}/*****
```

```
AIOPortInfo(int portChoice,
             int *hardwareType,
             int *boardNumber,
             int *portNumber)
```

```
Description:
```

```
The routine returns the AIO information of the port passed in portChoice.
```

```
Input:
```

```
portChoice
```

```
- port to return data about
```

```
hardwareType
```

```
- where to return hardware type
```

```
boardNumber
```

```
- where to return board number
```

```
portNumber
```

```
- where to return port number
```

```
Output:
```

```
hardwareType, boardNumber and portNumber filled in if successful
```

```
Returns:
```

```
0 if successful
```

```
*****/
```

```
int AIOPortInfo(int portChoice,
```

```
int *hardwareType,
```

```
int *boardNumber,
```

```
int *portNumber)
```

```
{
    int ccode;
```

```
AIOPORTINFO portInfo;
```

```
AIOPORTSEARCH portSearch;
```

```
while (AIOGetDriverList(hardware, &dvr) == AIO_SUCCESS)
```

```
{
    AIOBOARDLIST brd;
```

```
portInfo.returnLength = sizeof (AIOPORTINFO);
```

```
ccode = AIOGetFirstPortInfo(-1, -1, -1, &portSearch, &portInfo,
                             NULL, NULL, portOwner);
```

```
if (ccode)
    return (-1);
```

```
while (!ccode)
```

```
{
    if (portInfo.portNumber == portChoice)
```

```
    {
        if (portInfo.availability == AIO_AVAILABLE_FOR_ACQUIRE)
```

```
        {
            *hardwareType = portInfo.hardwareType;
```

```
*boardNumber = portInfo.boardNumber;
```

```
*portNumber = portInfo.portNumber;
```

```
return 0;
```

```
    }
```

```
    else
```

```
    {
        return (-2);
```

```
    }
```

```
    }
    ccode = AIOGetNextPortInfo(&portSearch, &portInfo, NULL, NULL,
                                portOwner);
```

```
return -1;
```

```
}/*****
```

```
InitializeAIO(void)
```

```
Description:
```

```
Initialize AIO. If it didn't initialize, AIOPortHandle will still be negative.
```

```
Input:
```

```
nothing
```

```
Output:
```

```
nothing
```

```
Returns:
```

```
nothing
```

```
*****/
```

```
void InitializeAIO(void)
```

```
{
    int ccode;
```

```
int hardware = AIO_HARDWARE_TYPE_WILDCARD;
```

```
int port = AIO_PORT_NUMBER_WILDCARD;
```

```
int board;
```

```
AIODRIVERLIST dvr;
```

```
dvr.returnLength = sizeof (AIODRIVERLIST);
```

```
if (AIOPortHandle == 0)
```

```
return;
```

```
while (AIOGetDriverList(hardware, &dvr) == AIO_SUCCESS)
```

```
{
    AIOBOARDLIST brd;
```



```

DloCallBackIndex = 0;
DloCallBackEscape = 0;
DloCallBackStarted = 0;
return(0);
}

/*****
*
* ValidPacket(BYTE *buf_to_rx,
*             int *len_to_rx)
*
* Description:
*   This routine validates a response from the modem. It checks the
*   length, checksum, opcode and status.
*
* Input:
*   buf_to_rx
*   a message
*   len_to_rx
*   length of the message
*
* Output:
*   len_to_rx
*   ze of the message
*   t the header
*
* Returns:
*   TRUE if packet is valid
*
* *****/
int ValidExplicitPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    LroEspkt_t *msg;
    WORD frameCrc = 0;
    WORD slipCrc = 0;
    WORD frameLen = 0;
    WORD crcVal = 0xffff;

    msg = (LroEspkt_t *)buf_to_rx;
    frameLen = (msg->length) & 0x7fff;

    if ((*len_to_rx - sizeof(WORD)) == frameLen)
    {
        CMovB(&buf_to_rx[frameLen], (BYTE *)&frameCrc, sizeof(frameCrc));

        slipCrc = calcCrc(crcVal, buf_to_rx, frameLen);
        if (slipCrc == frameCrc)
        {
            return(1);
        }
    }

    return(0);
}

int ValidPacket(BYTE *buf_to_rx, int *len_to_rx)
{
    unsigned short frameCrc=0; // CRC value contained in the frame
    unsigned short frameLen=0; // Length value contained in the frame
    unsigned short slipCrc=0; // CRC returned from calculation
    LroAsyncMsg_t *msg;

```

```

DcauResponse_t *d_msg;
int status = FALSE;

```

```

msg = (LroAsyncMsg_t *)buf_to_rx;
// extract the frame length contained in the first 2 bytes of the frame
frameLen = msg->header.length;
frameLen &= 0x7fff;

```

```

if ((*len_to_rx - sizeof(unsigned short)) == frameLen)
{

```

```

    // since sliplen includes CRC
    // extract the crc contained in the last 2 bytes of the frame
    frameCrc = msg->data[frameLen - LRO_ASYNC_HDR_SIZE];
    frameCrc += msg->data[frameLen - LRO_ASYNC_HDR_SIZE + 1] * 256;

    slipCrc = calcCrc (INITCRC, (unsigned char *) buf_to_rx, frameLen);
    if (slipCrc == frameCrc)
    {

```

```

        d_msg = (DcauResponse_t *) (msg->data);
        if (d_msg->opcode == 1 && d_msg->status == 0)
        {
            *len_to_rx = frameLen - RETURN_POINT;
            status = TRUE;
        }
    }
}

```

```

return status;
}

```

```

DloCallBackRead(void)

```

```

Description:

```

```

    This routine reads as many bytes as it can from the modem
    on behalf of the process that scheduled a receive thru
    DloScheduleReceive(). It's called by the main DLO thread
    as long as a call back is scheduled(DloCallBack != 0) and
    the modem has sent the previous request. All Slip specific
    characters are stripped and all Slip escape characters are
    converted back to ASCII. If the end of the message is hit,
    call the processes event routine with the message.

```

```

Input:    nothing

```

```

Output:   nothing

```

```

Returns:  nothing

```

```

static void DloCallBackRead()
{

```

```

    BYTE    value;

```

```

    for(;;)
    {

```

```

        if (DloReceive(&value, 1) == 0)
            break;

```

```

        if (DebugFlag)
            putchar(value);
    }
}

```



```

void DioMain(void *parm)
{
    LONG curticks, delta;
    LONG count, bytesRead;
    WORD state;
    BYTE value;
    int event, eventLen;
    char *pPtrAtBegin, *pPtrAtEnd;
    LONG extStatus;

    parm = parm;
    DioLastKnownTickCount = GetCurrentTime();
    while(!ExitingFlag)
    {
        delay(1000 / 6);
        count = 0;
        bytesRead = 0;
        if (AIOGetPortStatus(AIOPortHandle,
            &count, /* write count */
            0, /* write status */
            &bytesRead, /* read count */
            0, /* read status */
            &extStatus,
            0) == 0)
        {
            extStatus &= AIO_EXTSTA_DCD;
            if (extStatus != DioLastDCD)
            {
                if (!extStatus)
                    StateMachine(DLOE_DISCONN);
                DioLastDCD = extStatus;
            }
        }
        UpdateModemLights(count, bytesRead, DioLastDCD);
        if (DloState == DLOS_IDLE)
        {
            if (DloPXMitCount)
            {
                DloConn = DLO_CONN_PACKAGE;
                StateMachine(DLOE_SEND);
            }
            else if (DloIXmitCount)
            {
                DloConn = DLO_CONN_INET;
                StateMachine(DLOE_SEND);
            }
        }
        curticks = GetCurrentTime();
        if (curticks < DioLastKnownTickCount)
        {
            delta = curticks + (0xffffffff - DioLastKnownTickCount);
        }
        else
        {
            delta = curticks - DioLastKnownTickCount;
        }
        DioLastKnownTickCount = curticks;
    }
}

```

```

if (DIOStats && curticks > DPCNextRegistrationCheck)
{
    char buf[16];
    LONG hw;
    double key;
    CustVars* customPtr = (CustVars*)&DIOStats->CustomVaria
bleCount);
    LONG rxFreq = customPtr->CustomVariable[1] / 10;
    DPCSetMaxConnections((LONG*)&key);
    if (strcmp(DloCfg.base_license, "Helius, Inc.", 8) == 0)
    {
        DPCNextRegistrationCheck = (LONG){-1};
        goto skipRegistrationCheck;
    }
    /* get hardware serial number */
    *buf = 0;
    DIOGetSN(buf);
    hw = strtoul(buf, 0, 10);
    if (hw == 0) {
        ConsolePrintf("\r\nDPCAgent: could not obtain hardware
serial number of DPC card\n");
        goto disabledDPCAgent;
    }
    /* compute the registration key */
    if (DebugFlag == 0x98) {
        printf("\rRegCheck: %e\n", key);
        HexAsciiDump((void*)&key, sizeof(key));
    }
    key *= hw + DPC_IP_Address;
    if (DebugFlag == 0x98) {
        printf("\rRegCheck: %e %d %08x\n",
            key, hw, DPC_IP_Address);
        HexAsciiDump((void*)&key, sizeof(key));
    }
    if (key == *(double*)&DloCfg.key)
        DPCNextRegistrationCheck = curticks + 131072; /*
120 minutes */
    else
    {
        ConsolePrintf("\r\nDPCAgent: detected a
disabledDPCAgent:
        DPCMaxConnections = 3;
        RingTheBell();
        if (DPCNextRegistrationCheck) {
            DloCfg.gateway_address = 0;
            StateMachine(DLOE_TIMEOUT);
            DPCNextRegistrationCheck = curticks + 2185; /*
2 minutes */
        }
        else
        {
            DPCNextRegistrationCheck = curticks + 131072;
        }
        /* 120 minutes */
    }
    if (DPCNextRegistrationCheck & 0xffffffffE0) == 0xffffffff
    {
        DPCNextRegistrationCheck += 17; /* wrap */
    }
    skipRegistrationCheck:
    if (AIOPortHandle >= 0)
    {

```

```

for (;;)
{
    || count == 0)

    if (AIOReadStatus(AIOPortHandle, &count, &state)
        break;
    AIOReadData(AIOPortHandle, &value, 1, &bytesRead);
    if (DebugFlag && DloState != DLOS_CONN)
        putchar(value);

    if (DloRcvCount < DLOBUFSIZE)
    {
        DloRcvBuffer[DloRcvIndex] = value;
        DloRcvIndex++;
        if (DloRcvIndex >= DLOBUFSIZE)
            DloRcvIndex = 0;
        DloRcvBuffer[DloRcvIndex] = 0;
        DloRcvCount++;
    }
    if (value != '\n' && value != '\r')
    {
        memcpy(DloCommandBuffer, DloCommandBuffere
        DloCommandBuffer[DLOCMDBUFSIZE - 1] = va
        if (DloCommandIndex < DLOCMDBUFSIZE)
            DloCommandIndex++;
    }
    if (value != '\r')
        continue;

    pPtrAtBegin = DloCommandBuffer + DLOCMDBUFSIZE -
    eventLen = strlen(DloCompareString(DLOE_CONNECT)
    if (strncmp(pPtrAtBegin, DloCompareString(DLOE_C
    {
        memcpy(ConnectBaudStr, pPtrAtBegin + event
        ConnectBaudStr[DloCommandIndex - eventLe
        DloFlushReceive();
        StateMachine(DLOE_CONNECT);
    }
    else
    {
        for (event = 0; event < DLOENUM ; event+
        {
            if (event == DLOE_CONNECT)
                continue;

            eventLen = CStrLen(DloCompareStr
            if (eventLen == 0)
                continue;

            pPtrAtEnd = DloCommandBuffer + D
            if (strncmp(pPtrAtEnd, DloCompar
                continue;

            DloFlushReceive();
            StateMachine(event);
        }
    }

    LOCMDBUFSIZE - eventLen;
    eString(event), eventLen) != 0)

    if (DloReadStatus(AIOPortHandle, &count, &state)
        break;
    DloCommandIndex = 0;

    if (delta)
    {
        /* Check DLO state machine timer.
        */
        if (DloTimer)
        {
            if (delta >= DloTimer)
            {
                DloTimer = 0;
                StateMachine(DLOE_TIMEOUT);
            }
            else
            {
                DloTimer -= delta;
            }
        }
        /* Check Packet lifetime timer.
        * This timer is initialized when data is added to the
        * data buffer via DloSend.
        * If the timer expires, clear the data buffer.
        */
        if (DloPacketLifetime)
        {
            if (delta >= DloPacketLifetime)
            {
                DloPacketLifetime = 0;
                if (DloConn == DLO_CONN_PACKAGE)
                    DloPxmItCount = 0;
                else
                    DloIXmitCount = 0;
            }
            else
            {
                DloPacketLifetime -= delta;
            }
        }
        /* Check the Receive Call Back timer.
        * If it times out, call call back routine with timeout
        */
        if (DloCallBack)
        {
            if (DloCallBackWait)
            {
                /* Wait until the send request is sent b
                if (DloEmpty())
                {
                    DloCallBackWait = 0;
                }
            }
            else
            {
                before starting timer */
            }
        }
    }
}

```

```
timeout flag set */
loCallBackIndex, 1);

        if (delta >= DioCallBackTimeout)
        {
            /* Timeout!!! Call routine with
            DioCallBackWait = 0;
            DioCallBack(DioCallBackBuffer, D
            DioCallBack = 0;
            }
            else
            DioCallBackTimeout -= delta;
            /* Read any available modem characters f
            DioCallBackRead();
        }
    }
    if (AIOPortHandle >= 0)
        AIOReleasePort(AIOPortHandle);
    DPCModemPID = 0;
}
```



```

;
INIT equ 0
SYNTH_PRGM equ 1
ACQ_PD_DELAY equ 2
ACQ_PD equ 3
ENABLE_BTR equ 4
START_SEARCH_FOR_FEC equ 5
CHECK_FOR_FEC_LOCK equ 6
SET_OTHER_MODE equ 7
TRACKING equ 8
POINTING_ACQ equ 9
POINTING_TRACKING equ 10
HALT equ 11

; New Stuff DBS
; demod command definitions
;
ACQUIRE_MODE equ 0
HALT_MODE equ 2
BUSY_MODE equ 3
POINTING_MODE equ 4

; /** start a new acquisition
; /** Do nothing
; /** Trying to acquire
; /** Special test mode
;

DEFAULT_RX_FREQ equ 1330
BIT_OFF equ 00h

; BtrControlAddr bits - write register 0
;
FREQ_PWR_MASK equ 0E0h
PHASE_PWR_MASK equ 07h
BTR_SENSE_MASK equ 08h
BTR_ERR_ENA_MASK equ 10h
FREQ_PWR_OFFSET equ 20h
PHASE_PWR_OFFSET equ 01h

; AfcControlAddr bits - write register 1
;
SWP_ENA_MASK equ 01h
SWEEP_DIR_SENSE_MASK equ 08h
AFC_SENSE_MASK equ 10h
EXT_INT_MASK equ 20h
BPSK_MASK equ 40h
ROM_ENA_MASK equ 80h
SQF_PEAK_EN_MASK equ 02h

; BitDetControlAddr bits - write register 2
;
SOFT_THRS_MASK equ 1Fh
DECODER_INFC_SEL_MASK equ 80h
VIT_SEQ_MASK equ 40h
SOFT_THRS_OFFSET equ 01h

; AgcFirControlAddr bits - write register 3
;
AGC_REF_MASK equ 1Fh
AGC_SENSE_MASK equ 40h
FIR_BYPASS_MASK equ 80h
SWAP_IQ_MASK equ 20h
AGC_RBF_OFFSET equ 01h

; CrlkControlAddr bits - write register B
;
CRLK_DET_PWR_MASK equ 07h
CRLK_FC_MASK equ 38h
CRLK_GAIN_MASK equ C0h

; CRLK_DET_PWR_OFFSET equ 01h
; CRLK_FC_OFFSET equ 08h
; CRLK_GAIN_OFFSET equ 40h

; SynthSerControlAddr bits - write register C
;
SDATA_MASK equ 01h
SCLK_MASK equ 02h
SENA_MASK equ 04h
MODE_MASK equ 08h
CRL_ACC_ENABLE equ 10h
DEPUNC_BYPASS_MASK equ 20h
RESET_FEC_ACQ_MASK equ 40h
RESET_BTR_ACC_MASK equ 80h

; DaadOffsetControlAddr bits - write register E
;
I_CHANNEL_OFFSET equ 01h
CRL_ERROR_OFFSET equ 08h
BTR_ERROR_OFFSET equ 40h

;
;
SYNTH_LOCK_MASK equ 01h
CRL_LOCK_MASK equ 02h
SWEEPING_MASK equ 04h
DIR_MASK equ 08h
FEC_LOCK_MASK equ 10h
TUNER_TYPE_MASK equ 20h
TUNER_TYPE_2_MASK equ 08h
VENDOR_ID_MASK equ 0F0h

; /** Frequency offsets */
;
PLUS_OFFSET equ 40
ZERO_OFFSET equ 0
MINUS_OFFSET equ -40
OFFSET_THRESHOLD equ 1152
FREQ_BASE equ 9011
K_TRACK equ 1736
K_REACQ equ 29622
NOM_COUNT_TRACK equ 48761
NOM_COUNT_REACQ equ 19432
SYNTH_CHANNEL_SIZE equ 3645
SYNTH_RATIO equ 64
SYNTH_CHANNELS_PER_STEP equ 40
SYNTH_FIRST_CHANNEL equ 3788

; BPSK equ BPSK_MASK OR ROM_ENA_MASK
; /** Possible Viterbi modes */
;
LOWRATE equ 0
HIGHRATE equ 1

; /** demod status states */
;
UNLOCKED equ 0
LOCKED equ 1

; Configuration equates
;
RBD_BASE_ADDR equ 0a000h
ADAP_RBD_NUM equ 128
RBD_BUFFER_SIZE equ 1024
LOCAL_BUF_NUM equ 400

; /** for SHARP type tuners */

```

```

; RBD status bits.
;
; FRAMING_ERR      equ 0001h      ; Framing error
; CRC_ERR          equ 0002h      ; CRC error
; ABORT            equ 0004h      ; Abort
; ALIGN_ERR        equ 0008h      ; Alignment error
; DES_ERR          equ 0010h      ; DES Error
; SOF_BIT          equ 0020h      ; Start of Frame (not used)
; EOF_BIT          equ 0040h      ; End of Frame
; OVERRUN_ERR      equ 0080h      ; Frame Overrun bit
; EMPTY           equ 8000h
; STATUS_ERROR     equ FRAMING_ERR OR CRC_ERR OR ABORT OR ALIGN_ERR OR
; DES_ERR OR OVERRUN_ERR

```

```
DebugMessage macro mask, message
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx
```

```
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (2 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
```

```
DebugMessageExit:
    endm
```

```
DebugMessage1 macro mask, message, parm0
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx
```

```
    push parm0
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (3 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
```

```
DebugMessageExit:
    endm
```

```
DebugMessage2 macro mask, message, parm0, parm1
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx
```

```
    push parm1
    push parm0
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (4 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
```

```
DebugMessageExit:
    endm
```

```
DebugMessage3 macro mask, message, parm0, parm1, parm2
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx
```

```
    push parm2
    push parm1
    push parm0
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (5 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
```

```
DebugMessageExit:
    endm
```

```
DebugMessage4 macro mask, message, parm0, parm1, parm2, parm3
```

```
    local DebugMessageExit
```

```
    test DebugMask, mask
    je DebugMessageExit
```

```
    push eax
    push ecx
    push edx
```

```
    push parm3
    push parm2
    push parm1
    push parm0
    push offset message
    push DPCScreen
    call OutputToScreen
    lea esp, [esp + (6 * 4)]
```

```
    pop edx
    pop ecx
    pop eax
```

DebugMessageExit:

endm

DebugMessage5 macro mask, message, parm0, parm1, parm2, parm3, parm4

local DebugMessageExit

test DebugMask, mask
je DebugMessageExitpush eax
push ecx
push edx

push parm4

push parm3

push parm2

push parm1

push parm0

push offset message

push DPSCScreen

call OutputToScreen

lea esp, [esp + (7 * 4)]

pop edx

pop ecx

pop eax

DebugMessageExit:

endm

DebugMessage6 macro mask, message, parm0, parm1, parm2, parm3, parm4, parm5

local DebugMessageExit

test DebugMask, mask
je DebugMessageExit

push eax

push ecx

push edx

xor eax, eax

mov al, parm5

push eax

mov al, parm4

push eax

mov al, parm3

push eax

mov al, parm2

push eax

push parm1

mov al, parm0

push eax

push offset message

push DPSCScreen

call OutputToScreen

lea esp, [esp + (8 * 4)]

pop edx

pop ecx

pop eax

DebugMessageExit:

endm

SLOW macro

push eax
in al, 61h
in al, 61h
in al, 61h
pop eax
endm

BufferStruct struc

BufPtr dd 0 ; Pointer to buffer
DataSize dd 0 ; Size of buffer
BufferStruct ends

LAST_EOF equ 80000000h ; or with DataSize

MAX_APPL_RBD equ 350

MAX_CHAN equ 10

RBD_NOT_USED equ -1

MAX_CONF_ADDR equ 8

MAX_ADDR equ 16

RX_CNTL struc

RxChannel dd 0 ; Channel ID

RxESR dd 0 ; ESR to call when a packet

RX_CNTL ends ; is received.

RX_FLAG_STATS_ONLY equ 1

ChannelConfig struc

CfgChannel dd 0 ; Channel ID

CfgESR dd 0 ; max # of buffers used

CfgNumAddresses dd 0 ; Number of filter addrs

CfgAddress dd 6 dup (0) ; that is to follow

CfgGroupKey db 8 dup (0) ; list of addrs

CfgElementKey db 8 dup (0) ; Group keys

ChannelConfig ends ; Element keys

FilterStruct struc

FilterAddress db 6 dup (0) ; 48 bit address

FilterChannel db 2 dup (0) ; Align next field

FilterCmdBlkIndex dd 0 ; Channel ID

FilterTotalCount dd 0 ; RISC SW cmd blk index

FilterSeqCount dd 0 ; # of frames rxed

FilterSeqNum dd 0 ; # of out-of-seq frames

FilterStruct ends ; next seq expected

EbikStruct struc

EbikCmd dw 0 ;

EbikPortID dw 0 ;

EbikNotUsed dw 0 ;

EbikAddress db 6 dup (0) ;

EbikGroupKey db 8 dup (0) ;

EbikElemKey db 8 dup (0) ;

EbikStruct ends ;


```
db 'Mips Rx disable commands', 0
db 'Mips frames accepted', 0
db 'Mips frames rejected from no filter match', 0
db 'Mips frames rejected from zero address', 0
db 'Mips frames rejected from adapter disabled', 0
db 'Mips frames rejected from low buffer pool', 0

db 0, 0

EndOfStrings equ $
; .....
; Driver Parameter Block to pass to MSM.
; .....
; align 4
DriverParameterBlock
DriverParameterSize dd DriverParameterBlockSize
DriverStackPointer dd 0
DriverModuleHandle dd 0
DriverBoardPointer dd 0
DriverAdapterPointer dd 0
DriverConfigTemplatePtr dd DriverConfigTemplate
DriverFirmwareSize dd 0
DriverFirmwareBuffer dd 0
DriverNumKeywords dd 2
DriverKeywordText dd DPCKeywordText
DriverKeywordTextLen dd DPCTextLen
DriverProcessKeywordTab dd DPCProcessKeywordTab
DriverAdapterDataSpaceSize dd SIZE DriverAdapterDataSpace
DriverAdapterTemplate dd DriverAdapterDataSpaceTemplate
DriverStatisticsTable dd StatisticsVersion
DriverEndOfChainFlag dd 0
DriverMaxMulticast dd 0
DriverSendWantsECBs dd -1
DriverAESPtr dd 0
DriverNeedsBelow16Meg dd 0
DriverCallBackPtr dd offset DriverCallBack
DriverISRPtr dd offset DriverISR
DriverMulticastChangePtr dd offset DriverMulticastChange
DriverPollPtr dd 0
DriverResetPtr dd offset DriverReset
DriverSendPtr dd offset DriverSend
DriverShutdownPtr dd offset DriverShutdown
DriverTxTimeoutPtr dd 0
DriverPromiscuousChangePtr dd offset DriverPromiscuousChange
DriverStatisticsChangePtr dd offset RefreshMipsStats
DriverRxLookAheadChangePtr dd 0
DriverManagementPtr dd offset DriverManagement
DriverEnableInterruptPtr dd offset DriverEnableInterrupt
DriverDisableInterruptPtr dd offset DriverDisableInterrupt

DriverParameterBlockSize equ $ - DriverParameterBlock
; .....
; Driver Management Dispatch Jump Table.
; .....
ManagementJumpTable label dword
dd offset GetMipsStats
dd offset OpenChannel
dd offset CloseChannel
```

```
dd offset GetSN
dd offset SignText
dd offset Address
dd offset DeleteAddress
dd offset RegisterAgentSendRoutine
dd offset BadParametersExit
dd offset RegisterAgent
dd offset ReturnTCBCompleteRoutine
LastManagementFunction equ (($ - ManagementJumpTable) / 4) - 1
; .....
; Copy of Virtual Adapter Data area to be copied at
; initialization.
; .....
; DriverAdapterDataSpaceTemplate DriverAdapterDataSpace <>
DriverConfigTemplate db 'HardwareDriverMLID
Signature 01 ; [ebx].MLIDCFG_MajorVersion
12 ; [ebx].MLIDCFG_MinorVersion
6 dup (0ffh) ; [ebx].MLIDNodeAddress
0010010001001001001b ; [ebx].MLIDModeFlags
0000 ; [ebx].MLIDBoardNumber
0000 ; [ebx].MLIDBoardInstance
00000000 ; [ebx].MLIDMaxRecvSize
00000000 ; [ebx].MLIDMaxRecvSize
00000000 ; [ebx].MLIDRecvSize
00000000 ; [ebx].MLIDCardName
00000000 ; [ebx].MLIDShortName
00000000 ; [ebx].MLIDFrameType
0000 ; [ebx].MLIDReserved0
0000 ; [ebx].MLIDFrameID
0001 ; [ebx].MLIDTransportTime
000000000 ; [ebx].MLIDRouteHandler
10 ; [ebx].MLIDLineSpeed
0000 ; [ebx].MLIDLookAheadSize
8 dup (00h) ; [ebx].MLIDReserved
MLID_MAJOR_VERSION ; [ebx].MLIDMajor
MLID_MINOR_VERSION ; [ebx].MLIDMinor
rVersion db
rVersion db
0010b ; [ebx].MLIDFlags
0010 ; [ebx].MLIDSendRetries
00000000 ; [ebx].MLIDLink
0000 ; [ebx].MLIDSharingFlags
0000 ; [ebx].MLIDSlot
02c0h, 40h, 0, 0 ; [ebx].MLIDPortsAndLengths
00000000 ; [ebx].MLIDMemoryDecode0
0000 ; [ebx].MLIDLength0
00000000 ; [ebx].MLIDMemoryDecode1
0000 ; [ebx].MLIDLength1
03, 0ffh ; [ebx].MLIDInterrupt
0ffh, 0ffh ; [ebx].MLIDMAUAge
00000000 ; [ebx].MLIDResourceTag
00000000 ; [ebx].MLIDConfiguration
00000000 ; [ebx].MLIDCommandString
18 dup (0) ; [ebx].MLIDLogicalName
00000000 ; [ebx].MLIDLinearMemory0
00000000 ; [ebx].MLIDLinearMemory1
0000 ; [ebx].MLIDChannelNumber
6 dup (0) ; [ebx].MLIDIOReserved
DriverNICShortName, 'DPC'
Message
```


WaitForCommandDone:

```
    push    ecx
    mov     eax, [ebp].TimerTag
    push    eax
    push    2
    call    DelayMyself
    add     esp, (2 * 4)
    pop     ecx

    cli
    mov     edx, [ebp].IOMsgRamPtr
    mov     eax, (18100h + (14 * size EbldStruct)) / 2
    out     dx, ax

    mov     edx, [ebp].IOMsgRam
    in      ax, dx
    sti
    cmp     ax, 0eeh
    je      SignTextError

    cmp     ax, 11h
    je      ReadyToGetSignature
    dec     ecx
    jne     WaitForCommandDone
```

ReadyToGetSignature:

```
    mov     edi, [esi+8]
    cli

    mov     edx, [ebp].IOMsgRamPtr
    mov     eax, 18790h / 2
    out     dx, ax

    mov     ecx, 4
    mov     edx, [ebp].IOMsgRam
    GetsSignatureLoop:
        in      ax, dx
        mov     [edi], ax
        add     edi, 2
        dec     ecx
        jne     GetsSignatureLoop

        sti
        mov     dword ptr [esi+4], 8
        xor     eax, eax
        ret
```

SignTextError:

```
    mov     eax, -1
    mov     ret
```

SignText

```
    subttl  -- GetSN --
    page
```

;

; BEGIN_MANUAL_ENTRY(GetSN, DPC/API/GETSN)

; Name: GetSN

; Description: This routine is called by DriverManagement to
; return the adapters serial number.

; On Entry:

```
    EAX    N/A
    EBX    Frame Data Space
    ECX    N/A
    EDX    N/A
    EBP    Adapter Data Space
    ESI    ECB
    EDI    N/A
```

Note: Interrupts are in any state.

; On Return:

```
    EAX    Destroyed
    EBX    Preserved
    ECX    Destroyed
    EDX    Destroyed
    EBP    Preserved
    ESI    Preserved
    EDI    Preserved
```

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverManagement.
It is called at process time.

; See Also:

; END_MANUAL_ENTRY

;

public GetSN

GetSN proc

mov esi, [esi].RPacketOffset

cli

```
    mov     edx, [ebp].IOMsgRamPtr
    mov     eax, 18760h / 2
    out     dx, ax
```

mov ecx, 3

mov edx, [ebp].IOMsgRam

GetsNLoop:

```
        in      ax, dx
        mov     [esi], ax
        add     esi, 2
        dec     ecx
        jne     GetsNLoop
```

sti

mov [esi], cl

ret

GetSN endp

```
    subttl  -- CloseChannel --
    page
```

;

; BEGIN_MANUAL_ENTRY(CloseChannel, DPC/API/CLSCHAN)

; Name: CloseChannel

; Description: This routine is called by DriverManagement to

close the specified channel.

On Entry: EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI ECB
EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverManagement.
It is called at process time.

See Also:

END_MANUAL_ENTRY

public CloseChannel
CloseChannel proc

mov esi, [esi].RPacketOffset ; ESI -> config structure
mov esi, [esi] ; ESI = channel

cmp esi, MAX_CHAN
ja CloseChannelError
lea edi, [ebp].RxControl[esi*8]
cmp [edi].RxChannel, RBD_NOT_USED
je CloseChannelError

mov [edi].RxChannel, RBD_NOT_USED
mov [edi].RxESR, 0

mov ecx, MAX_ADDR
lea edi, [ebp].Filter

CloseChannelCloseFilterLoop:

cmp [edi].FilterChannel, esi
jne CloseChannelCloseFilterNext

mov [edi].FilterChannel, RBD_NOT_USED
mov eax, [edi].FilterCmdblkIndex
mov [ebp].EblkBusyFlags(eax), 0

push ecx
mov ecx, size EblkStruct
mul ecx
mov edx, [ebp].IOMsgRamPtr
add eax, 18100h
shr eax, 1
push eax
add eax, 3

out dx, ax

mov edx, [ebp].IOMsgRam
xor eax, eax
out dx, ax
out dx, ax
out dx, ax
out dx, ax

pop eax
pop ecx
mov edx, [ebp].IOMsgRamPtr
out dx, ax
mov edx, [ebp].IOMsgRam
mov eax, 1
out dx, ax

CloseChannelCloseFilterNext:

add edi, size FilterStruct
dec ecx
jne CloseChannelCloseFilterLoop

xor eax, eax
ret

CloseChannelError:

mov eax, -1
ret

CloseChannel endp
subttl -- Address --
page

BEGIN_MANUAL_ENTRY(Address, DPC/API/ADDRS)

Name: Address

Description: This routine is called by DriverManagement to
add the address passed in.

On Entry: EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI ECB
EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverManagement.
It is called at process time.

; See Also:

; END_MANUAL_ENTRY

;

public AddAddress
proc

```
mov esi, [esi].RPacketOffset
mov eax, [esi].CfgChannel
cmp ecx, MAX_CHAN
ja OpenChannelError

lea edi, [ebp].RxControl[eax*8]
cmp [edi].RxChannel, RBD_NOT_USED
je OpenChannelError
```

```
; Fall thru to AddAddr
;
```

AddAddress endp

```
subttl -- AddAddr --
page
```

;

; BEGIN_MANUAL_ENTRY(AddAddr, DPC/API/ADDADDR)

; Name: AddAddr

```
; Description: This routine is called by AddAddress and OpenChannel
; to add the address to the adapter.
```

; On Entry:

| | |
|-----|--------------------|
| EAX | N/A |
| EBX | Frame Data Space |
| ECX | N/A |
| EDX | N/A |
| EBP | Adapter Data Space |
| ESI | ECB |
| EDI | N/A |

```
; Note: Interrupts are in any state.
```

; On Return:

| | |
|-----|-----------|
| EAX | Destroyed |
| EBX | Preserved |
| ECX | Destroyed |
| EDX | Destroyed |
| EBP | Preserved |
| ESI | Preserved |
| EDI | Preserved |

; Flags:

```
; Note: Interrupts preserved.
```

```
; Remarks: This routine is called by AddAddress and OpenChannel.
; It is called at process time.
```

; See Also:

; END_MANUAL_ENTRY

;

public AddAddr

AddAddr proc

```
mov eax, dword ptr [esi].CfgAddress
DebugMessage1 DEBUG_IOCTL, AddAddrMsg, eax
```

```
mov ecx, [esi].CfgNumAddresses ; ECX = Number 0
```

```
f Addr
AddAddrLoop:
```

```
; First make sure this address is not a duplicate
;
```

```
lea edi, [ebp].Filter
mov edx, MAX_ADDR
AddAddrDuplicateLoop:
cmp [edi].FilterChannel, RBD_NOT_USED
je AddAddrDuplicateNext
mov eax, dword ptr [edi].FilterAddress
cmp eax, dword ptr [esi].CfgAddress
jne AddAddrDuplicateNext
mov ax, word ptr [edi].FilterAddress+4
cmp ax, word ptr [esi].CfgAddress+4
jne AddAddrDuplicateNext
```

```
mov eax, -1
ret
```

AddAddrDuplicateNext:

```
add edi, size FilterStruct
dec edx
jne AddAddrDuplicateLoop
```

```
; Find an empty slot in the filter table
;
```

```
lea edi, [ebp].Filter
xor edx, edx
AddAddrFindEmptyLoop:
cmp [edi].FilterChannel, RBD_NOT_USED
je AddAddrFindEmptyFound
add edi, size FilterStruct
inc edx
cmp edx, MAX_ADDR
jb AddAddrFindEmptyLoop
```

```
mov eax, -1
ret
```

AddAddrFindEmptyFound:

```
mov eax, [esi].CfgChannel
mov [edi].FilterChannel, eax
mov eax, dword ptr [esi].CfgAddress
mov dword ptr [edi].FilterAddress, eax
mov ax, word ptr [esi].CfgAddress+4
mov word ptr [edi].FilterAddress+4, ax
mov [edi].FilterTotalCount, 0
mov [edi].FilterSeqCount, 0
mov [edi].FilterSeqNum, 0
```

```
mov edx, 16
```

```
mov eax, 31
```

```
test [esi].CfgAddress+0, 02h
```

```
jnz AddAddrFindEblkLoop
```

```
mov edx, 2
```

```
mov eax, 13
```

```
AddAddrFindEblkLoop:
    cmp     [ebp].EblkBusyFlags[edx], 0
    je      AddAddrFindEblkFound
    inc     edx
    cmp     edx, eax
    jbe     AddAddrFindEblkLoop
    mov     eax, -1
    ret

AddAddrFindEblkFound:
    mov     [ebp].EblkBusyFlags[edx], 1
    mov     [edi].FilterCmdBlkIndex, edx
    mov     eax, [edi].FilterChannel
    lea     edi, [ebp].Eblk
    mov     [edi].EblkCmd, 0
    mov     [edi].EblkPortID, ax
    mov     ax, word ptr [esi].CfgAddress
    xchg    ah, al
    word ptr [edi].EblkAddress, ax
    mov     ax, word ptr [esi].CfgAddress+2
    xchg    ah, al
    word ptr [edi].EblkAddress+2, ax
    mov     edx, 16
    AddAddrBypass
    eax, dword ptr [esi].CfgGroupKey
    mov     dword ptr [edi].EblkGroupKey, eax
    mov     eax, dword ptr [esi].CfgGroupKey+4
    word ptr [edi].EblkGroupKey+4, eax
    mov     eax, dword ptr [esi].CfgElementKey
    word ptr [edi].EblkElementKey, eax
    mov     eax, dword ptr [esi].CfgElementKey+4
    word ptr [edi].EblkElementKey+4, eax
    mov     AddAddrBypass:
    pushfd
    cli
    push    ecx
    mov     eax, edx
    mov     ecx, size EblkStruct
    mul     ecx
    mov     edx, [ebp].IOMsgRamPtr
    shr     eax, 16100h
    mov     eax, 1
    push    esi
    push    dx, ax
    mov     edx, [ebp].IOMsgRam
    mov     ecx, size EblkStruct / 2
    mov     esi, edi
    cld
    AddAddrCopyEblk:
    lodsw
    out     dx, ax
    dec     ecx
    jne     AddAddrCopyEblk
    pop     esi
    pop     eax
    mov     ecx, [ebp].IOMsgRamPtr
    out     dx, ax
    mov     edx, [ebp].IOMsgRam
    mov     AddAddrCopyEblk:
    mov     esi, [esi].RPacketOffset
    mov     eax, [esi].CfgChannel
    cmp     eax, MAX_CHAN
    ja      DeleteAddressError
    lea     edi, [ebp].RxControl[esi*8]
    cmp     [edi].RxChannel, RBD_NOT_USED
    je      DeleteAddressError
    mov     ecx, [esi].CfgNumAddresses
```

```
out     dx, ax
popfd
```

```
dec     ecx
jne     AddAddrLoop
xor     eax, eax
ret
```

```
AddAddr endp
subttl  -- DeleteAddress --
page
```

```
*****
; BEGIN_MANUAL_ENTRY( DeleteAddress, DPC/API/DELADDR )
;
; Name: DeleteAddress
; Description: This routine is called by DriverManagement to
; delete the address passed in.
```

```
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI ECB
; EDI N/A
```

```
; Note: Interrupts are in any state.
```

```
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
```

```
; Flags:
```

```
; Note: Interrupts preserved.
```

```
; Remarks: This routine is called by DriverManagement.
; It is called at process time.
```

```
; See Also:
```

```
; END_MANUAL_ENTRY
```

```
*****
```

```
public DeleteAddress
DeleteAddress proc
```

```
    mov     esi, [esi].RPacketOffset
    mov     eax, [esi].CfgChannel
    cmp     eax, MAX_CHAN
    ja      DeleteAddressError
    lea     edi, [ebp].RxControl[esi*8]
    cmp     [edi].RxChannel, RBD_NOT_USED
    je      DeleteAddressError
    mov     ecx, [esi].CfgNumAddresses
```

```

cmp     ecx, MAX_CONF_ADDR
ja      DeleteAddressError

lea     ebx, [esi].CfgAddress
DeleteAddressLoop:

mov     ch, MAX_ADDR
lea     edi, (ebp).Filter
DeleteAddressFilterLoop:
mov     eax, dword ptr [ebx+0]
cmp     eax, dword ptr [edi].FilterAddress+0
jne     DeleteAddressNextFilter
mov     ax, word ptr [ebx+4]
cmp     ax, word ptr [edi].FilterAddress+4
jne     DeleteAddressNextFilter

mov     eax, [esi].CfgChannel
cmp     eax, [edi].FilterChannel
jne     DeleteAddressNextFilter

mov     [edi].FilterChannel, RBD_NOT_USED
mov     eax, [edi].FilterCmdblkIndex
mov     [ebp].EblksusyFlags[ebx], 0

pushfd
cli
push    ecx
mov     ecx, size EblkStruct
mul     ecx
mov     edx, [ebp].IOMsgRamPtr
add     eax, 18100h
shr     eax, 1
add     eax, 3
push    eax
out     dx, ax
mov     edx, [ebp].IOMsgRam
xor     eax, eax
out     dx, ax
out     dx, ax
out     dx, ax

pop     eax
pop     ecx
sub     eax, 3
mov     edx, [ebp].IOMsgRamPtr
out     dx, ax
mov     edx, [ebp].IOMsgRam
mov     eax, 1
out     dx, ax
popfd

DeleteAddressNextFilter:
add     edi, size FilterStruct
dec     ch
jne     DeleteAddressFilterLoop

DeleteAddressNext:
add     ebx, 6
dec     cl
jne     DeleteAddressLoop

xor     eax, eax
ret

DeleteAddressError:
mov     eax, -1

```

```

DeleteAddress     endp
subttl    -- RegisterAgentSendRoutine --
page

;.....
;
; BEGIN_MANUAL_ENTRY( RegisterAgentSendRoutine, DPC/API/DELADDR )
;
; Name:      RegisterAgentSendRoutine
;
; Description: This routine is called by DriverManagement to
;              register a Slip Send routine. The first dword in the first
;              ECB fragment points to the Slip Send routine to use. To
;              deregister the send routine, set the dword to a NULL.
;
; On Entry:   EAX    N/A
;             EBX    Frame Data Space
;             ECX    N/A
;             EDX    N/A
;             EBP    Adapter Data Space
;             ESI    ECB
;             EDI    N/A
;
; Note:       Interrupts are in any state.
;
; On Return:  EAX    Destroyed
;             EBX    Preserved
;             ECX    Destroyed
;             EDX    Preserved
;             EBP    Preserved
;             ESI    Preserved
;             EDI    Preserved
;
; Flags:
;
; Note:       Interrupts preserved.
;
; Remarks:    This routine is called by DriverManagement.
;             It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
;.....
;
; RegisterAgentSendRoutine proc
;
;     mov     esi, [esi].RPacketOffset    ; ESI -> config structur
;
;     mov     eax, [esi]
;     EAX -> Slip Send Routine Address
;     mov     [ebp].AgentSendRoutine, eax    ; Save it for later
;     xor     eax, eax
;     ret
;
; RegisterAgentSendRoutine     endp
subttl    -- RegisterAgent --
page
;.....
;
; BEGIN_MANUAL_ENTRY( RegisterAgent, DPC/API/REGAG )
;

```

```
; Name: RegisterAgent
; Description: This routine is called by DriverManagement to
;               register package delivery/internet agent. The first dword
;               in the first ECB fragment points to the Remove routine
;               that we must call before we are removed.
;
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  ECB
;            EDI  N/A
;
; Note:  Interrupts are in any state.
;
; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
;
; Flags:
;
; Note:  Interrupts preserved.
;
; Remarks: This routine is called by DriverManagement.
;           It is called at process time.
;
; See Also:
;
; END_MANUAL_ENTRY
; ...../
RegisterAgent proc
    mov     esi, [esi].RPacketOffset ; ESI -> config structur
    mov     eax, [esi]
    mov     [ebp].AgentRemoveRoutine, eax ; Save it for later
    xor     eax, eax
    ret
RegisterAgent endp
    subttl -- ReturnTCBCompleteRoutine --
    page
; ...../
; BEGIN_MANUAL_ENTRY( ReturnTCBCompleteRoutine, DPC/API/RETADS )
; Name: ReturnTCBCompleteRoutine
; Description: This routine is called by DriverManagement to
;               return the TCB Complete routine. The first dword
;               in the first ECB fragment points to a LONG which we will
;               return with a pointer to the TCB complete routine.
;
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
```

```
mov     esi, [esp + Parm0]
mov     ebp, OurAdapterDataSpace
inc     [ebp].MSMTxFreeCount
call    EtherTSMFastSendComplete
test    [ebp].MSMStatusFlags, SHUTDOWN
jne     GetNextSendExit

GetNextSendLoop:
test    [ebp].MSMStatusFlags, TXQUEUED
jnz     short GetNextSendGetIt

GetNextSendExit:
CPop
ret
```

```
GetNextSendGetIt:
call    EtherTSMGetNextSend
jne     short GetNextSendExit
call    DriverSend
jmp     GetNextSendLoop
; See if we have any more
```

```
DriverTCBComplete      endp

ReturnTCBCompleteRoutine proc
; esi, [esi].RPacketOffset
; dword ptr [esi], offset DriverTCBComplete
; esi -> fragment
mov     esi, [esi].RPacketOffset
mov     dword ptr [esi], offset DriverTCBComplete
xor     eax, eax
ret
```

```
ReturnTCBCompleteRoutine endp
subttl -- OpenChannel --
page
; *****
; BEGIN_MANUAL_ENTRY( OpenChannel, DPC/API/OPENCHAN )
;
; Name:      OpenChannel
; Description: This routine is called by DriverManagement to
;              open a channel on the adapter to receive packets
;              from.
```

```
; On Entry:  EAX    N/A
;            EBX    Frame Data Space
;            ECX    N/A
;            EDX    N/A
;            EBP    Adapter Data Space
;            ESI    ECB
;            EDI    N/A
; Note:      Interrupts are in any state.
```

```
; On Return: EAX    Destroyed
;            EBX    Preserved
;            ECX    Destroyed
;            EDX    Destroyed
;            EBP    Preserved
;            ESI    Preserved
;            EDI    Preserved
;
; Flags:
```

```
; esi -> TCB
; ebp -> Adapter Data Space
; Add to send resources
; Give it back
; Don't get next send
; if we're shutting down.
```

See Also:

END_MANUAL_ENTRY

.....

```
public OpenChannel
proc
```

```
mov     esi, [esi].RPacketOffset
; Don't open if we don't have signal lock.
cmp     [ebp].SignalQuality, 200
jb      OpenChannelError
; Find an empty RxControl entry
```

```
xor     ecx, ecx
lea     edi, [ebp].RxControl
FindEmptyRBDLoop:
cmp     [edi].RxChannel, RBD_NOT_USED
jne     FindEmptyRBDNext
; Found one. Initialize it before adding addresses.
```

```
mov     [edi].RxChannel, ecx
mov     [esi].CfgChannel, ecx
mov     eax, [esi].CfgESR
mov     [edi].RxESR, eax
```

```
push    edi
call    AddAddr
pop      edi
or      eax, eax
jne     OpenChannelDidntAdd
ret
```

```
OpenChannelDidntAdd:
mov     [edi].RxChannel, RBD_NOT_USED
mov     [edi].RxESR, 0
mov     eax, -1
ret
```

```
FindEmptyRBDNext:
inc     ecx
add     edi, size RX_CNTL
cmp     ecx, MAX_CHAN
jb      FindEmptyRBDLoop
```

```
OpenChannelError:
mov     eax, -1
ret
```

```
OpenChannel      endp
subttl -- RefreshMipsStats --
page
```

.....

BEGIN_MANUAL_ENTRY(RefreshMipsStats, DPC/API/RFSHMIPS)

```
; Name: RefreshMipsStats
; Description: This routine is called by to read the Mips stats
; from the adapter and to store them into the Local
; Mips Stats structure.
; On Entry: EAX N/A
; EBX N/A
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Destroyed
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by GetMipsStats and
; DriverCallBack.
; It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; .....
RefreshMipsStats proc
    pushfd
    cli
    mov     edx, [ebp].IOMsgRamPtr
    mov     eax, 18700h / 2
    out     dx, ax
    mov     edx, [ebp].IOMsgRam
    lea     edi, [ebp].MipsRxEnables
    mov     ecx, (size StatsBlk) / 2
    cld
GetMipsStatsLoop:
    in     ax, dx
    stosw
    loop   GetMipsStatsLoop
    popfd
    xor     eax, eax
    ret
RefreshMipsStats endp
    subttl -- GetMipsStats --
    page
; ..... \
```

```
; BEGIN_MANUAL_ENTRY( GetMipsStats, DPC/API/GETMIPS )
; Name: GetMipsStats
; Description: This routine is called by DriverManagement to
; return the Mips statistics.
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI ECB
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverManagement.
; It is called at process time.
; See Also:
; END_MANUAL_ENTRY
; .....
GetMipsStats proc
    push     esi
    call     RefreshMipsStats
    pop      esi
    mov     edi, [esi].RPacketOffset
    lea     esi, [ebp].MipsRxEnables
    mov     ecx, (size StatsBlk) / 4
    cld
    rep     movsd
    xor     eax, eax
    ret
GetMipsStats endp
; .....
; BEGIN_MANUAL_ENTRY( DriverMulticastChange, DPC/API/MULTI )
; Name: DriverMulticastChange
; Description: This routine will modify the NIC's multicast registers to
; enable it to receive the multicast addresses listed in
; the multicast table. Each entry in the multicast table is as
; follows:
; ..... \
```

bytes 0-5 = Multicast Address.
bytes 6-7 = Entry used (Non zero if used).

On Entry: EAX N/A
EBX N/A
ECX # of Entries in Table(0 if empty)
EDX N/A
EBP @ Adapter Data Space
ESI @ Multicast Table
EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Destroyed
EDI Destroyed

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by the ethernet media module.
It can be called at process or interrupt time.

See Also: ETHERTSM\ETHERTSMDdmMulticastAddress
ETHERTSM\ETHERTSMDleteMulticastAddress
ETHERTSM\ETHERTSMDupdateMulticast

END_MANUAL_ENTRY

DriverMulticastChange proc

First reset Multicast Address Registers.

ret

DriverMulticastChange endp
subttl -- DriverPromiscuousChange --
page

BEGIN_MANUAL_ENTRY(DriverPromiscuousChange, DPC/API/PROMISCU)

Name: DriverPromiscuousChange

Description: This routine will enable/disable the Promiscuous Mode.

On Entry: EAX N/A
EBX N/A
ECX 0 to disable the Promiscuous mode
EDX N/A
EBP @ Adapter Data Space
ESI @ Multicast Table
EDI N/A

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Destroyed
EDI Destroyed

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by the ethernet media module.
It can be called at process or interrupt time.

See Also: ETHERTSM\ETHERTSMPromiscuousChange
END_MANUAL_ENTRY

DriverPromiscuousChange proc

ret

DriverPromiscuousChange endp
subttl -- CalculatedDriftDelta --
page

BEGIN_MANUAL_ENTRY(CalculatedDriftDelta, DPC/API/CALCDD)

Name: CalculatedDriftDelta

Description: Acquisition State Routine.

On Entry: EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI N/A
EDI N/A

Note: Interrupts are in any state.

On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by InitState.
It can be called at process or interrupt time.

See Also:


```
mov     [ebp].GLOffset, ecx
inc     eax
xor     edx, edx
mov     ecx, 3
div     ecx
mov     [ebp].GLOffset, edx
```

StepCalcRx:

```
mov     [ebp].Drift, 0
call    CalculateFreq
mov     [ebp].Drift, NOM_COUNT_TRACK
ret
```

```
Step    endp
subttl  -- InitState --
page
```

```
; .....
; BEGIN_MANUAL_ENTRY( InitState, DPC/API/INITSTA )
;
; Name:      InitState
; Description: Acquisition State Routine.
;
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  N/A
;            EDI  N/A
;
; Note:      Interrupts are in any state.
```

```
; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
```

Flags:

```
; Note:      Interrupts preserved.
```

```
; Remarks:   This routine is called by DriverCallBack.
;            It can be called at process or interrupt time.
```

```
; See Also:
```

```
; END_MANUAL_ENTRY
```

```
; .....
; public InitState
; proc
```

```
InitState
public InitState
proc
cmp     [ebp].TrackingMode, TRUE
jne     InitStateNotTracking
cmp     DebugMask, 0
jne     InitStateNoMsg
```

```
mov     eax, offset InitStateTrackMsg
cmp     InitStateTrackMsg
je       InitStateNoMsg
mov     LastDebugMessage, eax
push    eax
push    DPCScreen
call    OutputToScreen
lea     esp, [esp + (2 * 4)]
```

InitStateNoMsg:

```
call    CalculateDriftDelta
add     eax, NOM_COUNT_REACQ

mov     edx, [ebp].IOCountNomLowAddr
out     dx, al
shr     al, 8
mov     edx, [ebp].IOCountNomHighAddr
out     dx, al

mov     edx, [ebp].IOGateCountHighAddr
mov     eax, [ebp].ReacqGateCount
out     dx, al
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
or      al, RESET_FEC_ACQ_MASK
out     dx, al
```

```
mov     edx, [ebp].IOBtrControlAddr
xor     eax, eax
out     dx, ax
```

```
mov     edx, [ebp].IOAfcControlAddr
in      al, dx
or      al, SWP_ENA_MASK
out     dx, al
```

```
mov     edx, [ebp].IOBtrControlAddr
in      al, dx
or      al, FREQ_PMR_OFFSET OR 6 OR BTR_SENSE_MASK
out     dx, al
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
or      al, RESET_FEC_ACQ_MASK
out     dx, al
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
cmp     [ebp].ViterbiMode, LOWRATE
jne     InitStateSetMode
and     al, NOT MODE_MASK
jmp     InitStateCheckRate
```

```
InitStateSetMode:
or      al, MODE_MASK
InitStateCheckRate:
out     dx, al
```

```
mov     edx, [ebp].IOSpareIOControlAddr
al, 0ah
cmp     [ebp].ViterbiOnly, 2
jne     InitStateSetRate
mov     al, 0bh
cmp     [ebp].ViterbiOnly, 1
jne     InitStateSetRate
```

```

mov     al, 0fh
InitStateSetRate:
out     dx, al
mov     [ebp].NextState, ENABLE_BTR
jmp     InitStateCheckPointing

InitStateNotTracking:
cmp     DebugMask, 0
je      InitStateNNoMsg
mov     eax, offset InitStateNotTrackMsg
cmp     eax, LastDebugMessage
je      InitStateNNoMsg
mov     LastDebugMessage, eax
push    eax
push    DPCScreen
call    OutputToScreen
lea     esp, (esp + (2 * 4))

InitStateNNoMsg:
mov     edx, [ebp].IOBitDetControlAddr
xor     eax, eax
out     dx, al

mov     edx, [ebp].IOSpareIOControlAddr
mov     al, 0ah
cmp     [ebp].ViterbiOnly, 2
je      InitStateNTSetRate
mov     al, 0bh
cmp     [ebp].ViterbiOnly, 1
je      InitStateNTSetRate
mov     al, 0fh
InitStateNTSetRate:
out     dx, al

mov     edx, [ebp].IODaAdOffsetControlAddr
xor     eax, eax
out     dx, al

mov     edx, [ebp].IOAfcControlAddr
mov     eax, [ebp].ModulationScheme
or      eax, SWEEP_DIR_SENSE_MASK
out     dx, al

mov     edx, [ebp].IOSweepRateAddr
mov     al, 8ah
out     dx, al

mov     edx, [ebp].IORateCountHighAddr
mov     eax, [ebp].RateCount
out     dx, al

mov     edx, [ebp].IOCountDeltaAddr
mov     eax, [ebp].SqfDeltaCount
out     dx, al

mov     eax, [ebp].NomCountSearch
[ebp].TuneCount, eax
mov     edx, [ebp].IOCountNomLowAddr
out     dx, al
shr     eax, 8
mov     edx, [ebp].IOCountNomHighAddr
out     dx, al

mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx

```

```

or      al, RESET_FEC_ACQ_MASK
out     dx, al

mov     edx, [ebp].IOBtrControlAddr
xor     eax, eax
out     dx, al

mov     edx, [ebp].IOAfcControlAddr
in      al, dx
or      al, SWP_ENA_MASK
out     dx, al

mov     edx, [ebp].IOAgcFirControlAddr
mov     al, 10 OR AGC_SENSE_MASK
out     dx, al

mov     edx, [ebp].IOBtrControlAddr
in      al, dx
or      al, FREQ_PWR_OFFSET OR 6 OR BTR_SENSE_MASK
out     dx, al

mov     edx, [ebp].IOCrikThrLowAddr
mov     al, 60h
out     dx, al

mov     edx, [ebp].IOCthAddr
mov     al, 0e0h
out     dx, al

mov     edx, [ebp].IOSynthSerControlAddr
mov     al, SENA_MASK
cmp     [ebp].ModulationScheme, BPSK
jne     InitStateSetSena
or      al, DEPUNC_BYPASS_MASK
InitStateSetSena:
out     dx, al

mov     edx, [ebp].IOCrikControlAddr
mov     al, 16 OR CRLK_GAIN_OFFSET OR CRLK_DET_PWR_OFFSET
out     dx, al

mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
cmp     [ebp].ViterbiMode, LOWRATE
jne     InitStateResetBtr
or      al, RESET_BTR_ACC_MASK
and     al, NOT MODE_MASK
jmp     InitStateClearBtr
InitStateResetBtr:
or      al, MODE_MASK OR RESET_BTR_ACC_MASK
InitStateClearBtr:
out     dx, al

in      al, dx
and     al, NOT RESET_BTR_ACC_MASK
out     dx, al

call    Step

mov     [ebp].NextState, ACQ_PD

InitStateCheckPointing:
mov     [ebp].RateCount, 0
mov     [ebp].PointingFlag, TRUE

```

```
cmp [ebp].DemodCommand, POINTING_MODE
je InitStateExit
mov [ebp].PointingFlag, FALSE
```

InitStateExit:

```
mov [ebp].CurrentState, SYNTH_PRGM
mov [ebp].SignalQuality, 0
mov [ebp].DemodCommand, BUSY_MODE
mov [ebp].DemodStatus, UNLOCKED
mov [ebp].FecStatus, UNLOCKED
ret
```

```
InitState endp
subttl -- ProgTuner --
page
```

```
; BEGIN_MANUAL_ENTRY( ProgTuner, DPC/API/PROGTUN )
```

```
; Name: ProgTuner
```

```
; Description: Acquisition State Routine.
```

```
; On Entry:
```

```
EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI N/A
EDI N/A
```

```
; Note: Interrupts are in any state.
```

```
; On Return:
```

```
EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved
```

```
; Flags:
```

```
; Note: Interrupts preserved.
```

```
; Remarks: This routine is called by Tune.
; It can be called at process or interrupt time.
```

```
; See Also:
```

```
; END_MANUAL_ENTRY
```

```
public ProgTuner
proc
```

```
; EAX = data
; ECX = len
```

```
dec ecx
mov edx, 1
shl edx, cl
mov ecx, edx
mov esi, eax
```

```
; ESI = Data
```

ProgTunerLoop:

```
jecxz ProgTunerExit
```

```
mov edx, [ebp].IOSynthSerControlAddr
in al, dx
test esi, ecx
je ProgTunerClear
or al, SDATA_MASK
and al, NOT SCLK_MASK
out dx, al
jmp ProgTunerDelay
```

```
ProgTunerClear:
and al, NOT (SCLK_MASK OR SDATA_MASK)
out dx, al
```

```
ProgTunerDelay:
shr ecx, 1
```

```
mov edx, [ebp].IOStatusAddr
in al, dx
in al, dx
```

```
mov edx, [ebp].IOSynthSerControlAddr
in al, dx
or al, SCLK_MASK
out dx, al
```

```
mov edx, [ebp].IOStatusAddr
in al, dx
in al, dx
```

```
jmp ProgTunerLoop
```

ProgTunerExit:

```
mov edx, [ebp].IOSynthSerControlAddr
in al, dx
and al, NOT SCLK_MASK
out dx, al
```

```
ret
```

```
ProgTuner endp
subttl -- Tune --
page
```

```
; BEGIN_MANUAL_ENTRY( Tune, DPC/API/TUNE )
```

```
; Name: Tune
```

```
; Description: Acquisition State Routine.
```

```
; On Entry:
```

```
EAX N/A
EBX Frame Data Space
ECX N/A
EDX N/A
EBP Adapter Data Space
ESI N/A
EDI N/A
```

```
; Note: Interrupts are in any state.
```

```
; On Return: EAX Destroyed
```

```

; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
;
; Flags:
;
; Note: Interrupts preserved.
;
; Remarks: This routine is called by SynthPrgmState.
; It can be called at process or interrupt time.
;
; See Also:
;
; END_MANUAL_ENTRY
;
; .....
```

```

;
; public Tune
; proc
;
; [ebp].TunerTypeFound, SHARP
; TuneSharpPan
; [ebp].TunerTypeFound, PANASONIC
; TuneSharpPan
; [ebp].TunerTypeFound, SHARP_CUSTOM
; TuneSharpCustom
; ret
```

```

TuneSharpPan:
; mov
; in
; or
; out
;
; mov
; in
; and
; out
```

```

; [ebp].TrackingMode, 0
; TuneSetNA
;
; mov
; in
; and
; out
```

```

; eax, 2ch
; [ebp].TunerTypeFound, SHARP
; TuneProgTuner
; mov
; eax, 0ech
;
; mov
; ecx, 8
; call
; ProgTuner
```

```

; mov
; in
; or
; out
;
; mov
; in
; al, dx
; or
; al, SENA_MASK
; out
;
; mov
; in
; al, dx
; in
; al, dx
```

```

; mov
; in
; and
; out
```

```

; mov
; eax, 28h OR 2000h
; mov
; ecx, 16
; call
; ProgTuner
```

```

; mov
; in
; or
; out
```

```

; mov
; in
; al, dx
; in
; al, dx
```

```

; TuneSetNA:
; mov
; in
; and
; out
```

```

; mov
; add
; xor
; mov
; div
; mov
; or
```

```

; mov
; ecx, 16
; call
; ProgTuner
```

```

; mov
; eax, edi
; mov
; ecx, 8
; call
; ProgTuner
```

```

; mov
; in
; or
; out
```

```

; ret
```

```

; TuneSharpCustom:
```

```

; mov
; in
; and
; out
```

```

; mov
; in
; and
; out
```

```

; mov
; eax, 50h OR 8001h
; mov
; ecx, 16
; call
; ProgTuner
```

```

; mov
; in
; al, dx
; or
; al, SENA_MASK
; out
; dx, al
```

```
mov     edx, [ebp].IOStatusAddr
in      al, dx
in      al, dx

mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
and     al, NOT SENA_MASK
out     dx, al

mov     eax, [ebp].ChannelNumber
add     eax, [ebp].GLDrift
xor     edx, edx
mov     ecx, SYNTH_RATIO
div     ecx, edx
mov     edi, edx

mov     ecx, 11
call    ProgTuner

mov     eax, edi
shl     eax, 1
mov     ecx, 9
call    ProgTuner

mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
or      al, SENA_MASK
out     dx, al

mov     edx, [ebp].IOStatusAddr
in      al, dx
in      al, dx

mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
and     al, NOT SENA_MASK
out     dx, al

ret

Tune    endp
subttl  page
.....
; BEGIN_MANUAL_ENTRY( SynthPrmState, DPC/API/SYNTHPS )
;
; Name:      SynthPrmState
; Description: Acquisition State Routine.
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  N/A
;            EDI  N/A
; Note:      Interrupts are in any state.
; On Return: EAX  Destroyed
;            ECX  Preserved
;            ECX  Destroyed
```

```
EDX  Destroyed
EBP  Preserved
ESI  Preserved
EDI  Preserved

Flags:

Note:  Interrupts preserved.

Remarks:  This routine is called by DriverCallBack.
           It can be called at process or interrupt time.

See Also:

; END_MANUAL_ENTRY
;.....
;
; public SynthPrmState
; SynthPrmState proc
;
;     cmp     DebugMask, 0
;     je      SynthPrmStateNoMsg
;     mov     eax, offset SynthPrmMsg
;     cmp     eax, LastDebugMessage
;     je      SynthPrmStateNoMsg
;     mov     LastDebugMessage, eax
;     push    eax
;     push    DPCScreen
;     call    OutputToScreen
;     lea     esp, [esp + (2 * 4)]
;     call    Tune
;     SynthPrmStateNoMsg:
;     call    Tune
;
;     mov     [ebp].TrackingMode, 0
;     cmp     [ebp].NextState, ACQ_PD
;     jne     SynthPrmClearT2
;
;     mov     [ebp].MaxSgf, 0
;     mov     [ebp].SgfAvg, 0
;     mov     [ebp].SgfWait, 0
;     mov     eax, [ebp].SgfCheckPoints
;     mov     [ebp].MaxCount, eax
;     mov     [ebp].T2Count, 60
;     mov     [ebp].CurrentState, ACQ_PD_DELAY
;     ret
;
; SynthPrmClearT2:
;     mov     [ebp].T2Count, 0
;     mov     [ebp].CurrentState, ACQ_PD_DELAY
;     ret
;
; SynthPrmState endp
; subttl  page
;.....
; BEGIN_MANUAL_ENTRY( AcqPDDelayState, DPC/API/ACQPDPS )
;
; Name:      AcqPDDelayState
; Description: Acquisition State Routine.
; On Entry:  EAX  N/A
;            EBX  Frame Data Space
```

```
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
;
; Note: Interrupts are in any state.
;
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
;
; Flags:
;
; Note: Interrupts preserved.
;
; Remarks: This routine is called by DriverCallBack.
; It can be called at process or interrupt time.
```

See Also:

END_MANUAL_ENTRY

```
public AcqPDDelayState
AcqPDDelayState proc
```

```
    cmp     DebugMask, 0
    je      AcqPDDelayStateNoMsg
    mov     eax, offset AcqPDDelayMsg
    cmp     eax, LastDebugMessage
    je      AcqPDDelayStateNoMsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPSCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
```

```
AcqPDDelayStateNoMsg:
    cmp     [ebp].T2Count, 0
    jne     AcqPDDelayExit
```

```
    mov     edx, [ebp].IOSweepRateAddr
    mov     al, 87h
    out     dx, al
```

```
    mov     edx, [ebp].IOAfcControlAddr
    in      al, dx
    and     al, NOT SQF_PEAK_EN_MASK
    out     dx, al
```

```
    mov     eax, [ebp].Sgfwait
    mov     [ebp].T2Count, eax
```

```
    mov     eax, [ebp].NextState
    mov     [ebp].CurrentState, eax
```

```
    mov     edx, [ebp].IOAfcControlAddr
    in      al, dx
    or      al, SQF_PEAK_EN_MASK
    out     dx, al
```

```
AcqPDDelayExit:
    ret
```

```
AcqPDDelayState endp
    subttl -- AcqPDSate --
page
```

```
; BEGIN_MANUAL_ENTRY( AcqPDSate, DPC/API/ACQPDS )
```

```
; Name: AcqPDSate
```

```
; Description: Acquisition State Routine.
```

```
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
```

```
; Note: Interrupts are in any state.
```

```
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
```

```
; Flags:
```

```
; Note: Interrupts preserved.
```

```
; Remarks: This routine is called by DriverCallBack.
; It can be called at process or interrupt time.
```

See Also:

END_MANUAL_ENTRY

```
public AcqPDSate
AcqPDSate proc
```

```
    cmp     DebugMask, 0
    je      AcqPDSateNoMsg
    mov     eax, offset AcqPDSMsg
    cmp     eax, LastDebugMessage
    je      AcqPDSateNoMsg
    mov     eax, LastDebugMessage
    push    eax
    push    DPSCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
```

```
AcqPDSateNoMsg:
    cmp     [ebp].T2Count, 0
    jne     AcqPDSExit
;
; xor     eax, eax
; mov     edx, [ebp].IOMaxSrfAddr
; in      al, dx
```

```

add     [ebp].SqfAvg, eax
cmp     eax, [ebp].MaxSqf
jbe     AcqPDDecMaxCount

mov     [ebp].MaxSqf, eax
mov     eax, [ebp].TuneCount
mov     [ebp].BestTuneCount, eax

AcqPDDecMaxCount:
dec     [ebp].MaxCount
jne     AcqPDMaxCountNotZero

mov     edx, [ebp].IOSweepRateAddr
mov     al, 8ah
out     dx, al

mov     edx, [ebp].IOCountNomLowAddr
mov     eax, [ebp].BestTuneCount
out     dx, al
shr     eax, 8
mov     edx, [ebp].IOCountNomHighAddr
out     dx, al

mov     edx, [ebp].IOCountDeltaAddr
mov     eax, [ebp].SqfDeltaCount
shl     eax, 1
out     dx, al

mov     eax, [ebp].SqfAvg
mov     ecx, [ebp].SqfCheckPoints
xor     edx, edx
div     ecx
add     eax, 2
mov     [ebp].SqfAvg, eax

mov     [ebp].T2Count, 40
mov     [ebp].NextState, ENABLE_BTR
mov     [ebp].CurrentState, ACQ_PD_DELAY

AcqPDExit:
ret

AcqPDMaxCountNotZero:
mov     eax, [ebp].TuneCount
add     eax, [ebp].SqfCheckStepSize
mov     [ebp].TuneCount, eax

mov     edx, [ebp].IOCountNomLowAddr
out     dx, al

mov     edx, [ebp].IOCountNomHighAddr
shr     eax, 8
out     dx, al

mov     [ebp].T2Count, 20
mov     [ebp].CurrentState, ACQ_PD_DELAY
ret

AcqPDState
subttl  -- EnableBTRState --
page
; .....
; BEGIN_MANUAL_ENTRY( EnableBTRState, DPC/API/ENBTRST )

```

```

; Name: EnableBTRState
; Description: Acquisition State Routine.
; On Entry: EAX N/A
;           EBX Frame Data Space
;           ECX N/A
;           EDX N/A
;           EBP Adapter Data Space
;           ESI N/A
;           EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
;            EBX Preserved
;            ECX Destroyed
;            EDX Destroyed
;            EBP Preserved
;            ESI Preserved
;            EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallback.
;           It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; .....
;
; public EnableBTRState
; EnableBTRState proc
;
;     cmp     DebugMask, 0
;     je      EnableBTRStateNoMsg
;     mov     eax, offset EnableBTRMsg
;     cmp     eax, LastDebugMessage
;     je      EnableBTRStateNoMsg
;     mov     LastDebugMessage, eax
;     push    eax
;     push    DPCScreen
;     call    OutputToScreen
;     lea     esp, [esp + (2 * 4)]
; EnableBTRStateNoMsg:
;     mov     edx, [ebp].IOBtrControlAddr
;     in      al, dx
;     or      al, BTR_ERR_ENA_MASK
;     out     dx, al
;
;     mov     [ebp].CurrentState, START_SEARCH_FOR_FEC
;     ret
;
; EnableBTRState endp
; subttl  -- StartSearchForFECState --
; page
; .....
; BEGIN_MANUAL_ENTRY( StartSearchForFECState, DPC/API/SRCIPEC )

```

```
; Name: StartSearchForFECState
; Description: Acquisition State Routine.
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallback.
; It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; .....
; public StartSearchForFECState
StartSearchForFECState proc
    cmp     DebugMask, 0
    je      StartSearchForFECStateNoMsg
    mov     eax, offset StartSearchForFECMsg
    cmp     eax, LastDebugMessage
    je      StartSearchForFECStateNoMsg
    mov     mov     LastDebugMessage, eax
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    cmp     SearchForFECStateNoMsg;
    je      SearchFECNotPointing
    mov     [ebp].CurrentState, POINTING_ACQ
    mov     eax, [ebp].SqfAvg
    add     eax, 2
    mov     [ebp].MaxSqf, eax
    jmp     SearchFECSetMax
SearchFECNotPointing:
    mov     [ebp].CurrentState, CHECK_FOR_FEC_LOCK
    mov     eax, [ebp].SqfAvg
    add     eax, 6
    mov     [ebp].MaxSqf, eax
SearchFECSetMax:
; Name: CheckforFECLockState
; Description: Acquisition State Routine.
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallback.
; It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; .....
; BEGIN_MANUAL_ENTRY( CheckforFECLockState, DPC/API/CHKFEC )
; Name: CheckforFECLockState
; Description: Acquisition State Routine.
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallback.
; It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; .....
; public CheckforFECLockState
public CheckforFECLockState
    mov     dx, al
    mov     [ebp].IOcthAddr
    and     al, NOT SWP_ENA_MASK
    out     dx, al
    mov     [ebp].IOSynthSerControlAddr
    in     al, dx
    or      al, RESET_FEC_ACQ_MASK
    out     dx, al
    mov     [ebp].TiCount, 300
    mov     [ebp].IOSynthSerControlAddr
    in     al, dx
    and     al, NOT RESET_FEC_ACQ_MASK
    out     dx, al
    ret
; StartSearchForFECState endp
; subttl -- CheckforFECLockState --
; page
; .....
; BEGIN_MANUAL_ENTRY( CheckforFECLockState, DPC/API/CHKFEC )
; Name: CheckforFECLockState
; Description: Acquisition State Routine.
; On Entry: EAX N/A
; EBX Frame Data Space
; ECX N/A
; EDX N/A
; EBP Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallback.
; It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; .....
; public CheckforFECLockState
public CheckforFECLockState
```

```

CheckForFEClockState proc
    cmp     DebugMask, 0
    je      CheckForFEClockStateNoMsg
    mov     eax, offset CheckForFEClockStateMsg
    cmp     eax, LastDebugMessage
    je      CheckForFEClockStateNoMsg
    mov     LastDebugMessage, eax
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, (esp + (2 * 4))

    CheckForFEClockStateNoMsg:
        mov     edx, [ebp].IOStatusAddr
        in     al, dx
        and     al, FEC_LOCK_MASK
        je      CheckFECNotLocked

        mov     [ebp].DemodStatus, LOCKED
        mov     [ebp].FecStatus, LOCKED

        mov     edx, [ebp].IOGateCountHighAddr
        xor     eax, eax
        out     dx, al

        mov     edx, [ebp].IOCountDeltaAddr
        mov     eax, [ebp].ReacqDeltaCount
        out     dx, al

        mov     edx, [ebp].IOBtrControlAddr
        in     al, dx
        and     al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)
        out     dx, al

        in     al, dx
        or      al, 5
        out     dx, al

        mov     [ebp].NextStepCount, 1
        mov     [ebp].T1Count, 500
        mov     [ebp].T2Count, 100
        mov     [ebp].CurrentState, TRACKING
        ret

CheckFECNotLocked:
    cmp     [ebp].T1Count, 0
    jne     CheckFECHaveT1Count

    mov     edx, [ebp].IOStatusAddr
    in     al, dx
    test    al, CRL_LOCK_MASK
    jne     CheckFECSetOtherMode

    mov     [ebp].CurrentState, INIT
    ret

CheckFECSetOtherMode:
    mov     [ebp].CurrentState, SET_OTHER_MODE

CheckFECExit:
    ret

CheckFECHaveT1Count:
    mov     edx, [ebp].IOStatusAddr
    in     al, dx

```

```

        test    al, CRL_LOCK_MASK
        jne     CheckFECExit
        mov     eax, [ebp].MaxSqr
        cmp     eax, [ebp].SqrAvg
        jbe     CheckFECExit
        sub     eax, 2
        mov     [ebp].MaxSqr, eax
        mov     edx, [ebp].IOChAddr
        out     dx, al
        ret

```

```

CheckForFEClockState endp
subttl -- SetOtherModeState --
page

```

```

; .....
; BEGIN_MANUAL_ENTRY( SetOtherModeState, DPC/API/SETOTHER )
;
; Name: SetOtherModeState
; Description: Acquisition State Routine.
; On Entry: EAX N/A
;           EBX Frame Data Space
;           ECX N/A
;           EDX N/A
;           EBP Adapter Data Space
;           ESI N/A
;           EDI N/A
; Note: Interrupts are in any state.
; On Return: EAX Destroyed
;            EBX Preserved
;            ECX Destroyed
;            EDX Destroyed
;            EBP Preserved
;            ESI Preserved
;            EDI Preserved
; Flags:
; Note: Interrupts preserved.
; Remarks: This routine is called by DriverCallback.
;          It can be called at process or interrupt time.
; See Also:
; END_MANUAL_ENTRY
; .....
; public SetOtherModeState
; SetOtherModeState proc

```

```

    cmp     DebugMask, 0
    je      SetOtherModeStateNoMsg
    mov     eax, offset SetOtherModeStateMsg
    cmp     eax, LastDebugMessage
    je      SetOtherModeStateNoMsg
    mov     LastDebugMessage, eax
    push    eax
    push    DPCScreen
    call    OutputToScreen

```

Thu Jul 17 14:46:01 1997

dpc.386

Page 59 of 174

dpc.386

Page 60

```
lea esp, [esp + (2 * 4)]
SetOtherModeStateNomsg:
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
cmp     [ebp].ViterbiMode, LOWRATE
jne     SetOtherModeToLow
```

```
or      al, MODE_MASK
out     dx, al
```

```
mov     [ebp].ViterbiMode, HIGHRATE
jmp     SetOtherModeIncRate
```

```
SetOtherModeToLow:
and     al, NOT MODE_MASK
out     dx, al
```

```
mov     [ebp].ViterbiMode, LOWRATE
```

```
SetOtherModeIncRate:
inc     [ebp].RateCount
```

```
cmp     [ebp].RateCount, 1
jg      SetOtherModeExit
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
or      al, RESET_FEC_ACQ_MASK
out     dx, al
```

```
mov     [ebp].TlCount, 180
```

```
mov     edx, [ebp].IOSynthSerControlAddr
in      al, dx
and     al, NOT RESET_FEC_ACQ_MASK
out     dx, al
```

```
mov     [ebp].CurrentState, CHECK_FOR_FEC_LOCK
ret
```

```
SetOtherModeExit:
mov     [ebp].CurrentState, INIT
ret
```

```
SetOtherModeState      endp
subttl  -- ReadWord --
page
```

```
REGIN_MANUAL_ENTRY( ReadWord, DPC/API/READWORD )
```

```
Name:      ReadWord
```

```
Description: Acquisition State Routine.
```

```
On Entry:  EAX  N/A
           EBX  Frame Data Space
           ECX  N/A
           EDX  N/A
           EBP  Adapter Data Space
           ESI  N/A
           EDI  N/A
```

```
Note:      Interrupts are in any state.
```

```
On Return:  EAX  Destroyed
           EBX  Preserved
           ECX  Destroyed
           EDX  Destroyed
           EBP  Preserved
           ESI  Preserved
           EDI  Preserved
```

```
Flags:
```

```
Note:      Interrupts preserved.
```

```
Remarks:   This routine is called by TrackingState,
           PointingAcquisitionState and PointingTrackingState
           It can be called at process or interrupt time.
```

```
See Also:
```

```
END_MANUAL_ENTRY
```

```
public ReadWord
proc
```

```
push     ebx
```

```
xor      ebx, ebx
mov      edx, edi
in      al, dx
mov      bh, al
```

```
mov      ecx, 4
ReadWordLoop:
mov      edx, esi
mov      in  al, dx
mov      mov  bl, al
```

```
mov      edx, edi
in      al, dx
cmp      al, bh
je       ReadWordExit
```

```
mov      bh, al
dec      ecx
jne      ReadWordLoop
```

```
ReadWordExit:
mov      eax, ebx
pop      ebx
ret
```

```
ReadWord      endp
subttl  -- TrackingState --
page
```

```
REGIN_MANUAL_ENTRY( TrackingState, DPC/API/TRCKST )
```

```
Name:      TrackingState
```

```
Description: Acquisition State Routine.
```

```

; On Entry:  EAX  N/A
;            EBX  Frame Data Space
;            ECX  N/A
;            EDX  N/A
;            EBP  Adapter Data Space
;            ESI  N/A
;            EDI  N/A

```

Note: Interrupts are in any state.

```

; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Destroyed
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved

```

Flags:

Note: Interrupts preserved.

Remarks: This routine is called by DriverCallback
It can be called at process or interrupt time.

See Also:

END_MANUAL_ENTRY

```

; .....
;
; public TrackingState
; TrackingState proc

```

```

    cmp     DebugMask, 0
    je      TrackingStateNoMsg
    mov     eax, offset TrackingStateMsg
    cmp     eax, LastDebugMessage
    je      TrackingStateNoMsg
    mov     eax, LastDebugMessage, eax
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]

```

TrackingStateNoMsg:

```

    mov     edx, [ebp].IOStatusAddr
    in      al, dx
    test    al, FEC_LOCK_MASK
    jne     TrackingStateReadQuality

```

```

    in      al, dx
    test    al, CRL_LOCK_MASK
    je      TrackingStateZero
    cmp     [ebp].T2Count, 0
    jne     TrackingStateT1

```

TrackingStateZero:

```

    mov     [ebp].TrackingMode, 0
    mov     [ebp].CurrentState, INIT
    ret

```

TrackingStateT1:

```

    mov     [ebp].T1Count, 1000
    ret

```

TrackingStateReadQuality:

```

    xor     eax, eax
    mov     edx, [ebp].IORelSgfAddr
    in      al, dx
    mov     [ebp].SignalQuality, eax

```

```

    cmp     DebugMask, 0
    je      SignalStrengthNoMsg
    cmp     LastSignalStrength, 0
    jne     SignalStrengthNoMsg

```

```

    mov     LastSignalStrength, 1
    cmp     eax, 200
    jb      SignalStrengthNone
    sub     eax, 200
    shl     eax, 1
    add     eax, 60
    jmp     SignalStrengthPrint

```

SignalStrengthNone:

```

    xor     eax, eax
    SignalStrengthPrint:
    push    eax
    push    offset SignalStrengthMsg
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (3 * 4)]

```

SignalStrengthNoMsg:

```

    cmp     [ebp].T1Count, 0
    jne     TrackingStateExit
    mov     [ebp].T1Count, 1000
    mov     [ebp].TrackingMode, 1

```

```

    mov     edi, [ebp].IOtuningHighAddr
    mov     esi, [ebp].IOtuningLowAddr
    call    ReadWord
    mov     [ebp].Drift, eax

```

```

    mov     ecx, 2
    cmp     eax, NOM_COUNT_TRACK + OFFSET_THRESHOLD
    ja      TrackingStateLocFound
    mov     ecx, 0
    cmp     eax, NOM_COUNT_TRACK - OFFSET_THRESHOLD
    jb      TrackingStateLocFound

```

TrackingStateLocFound:

```

    mov     [ebp].SearchLoc, ecx
    mov     [ebp].SearchLocFound, TRUE
    TrackingStateExit:
    ret

```

```

TrackingState endp
subttl -- PointingAcquisitionState --
page

```

```

; .....
;
; BEGIN_MANUAL_ENTRY( PointingAcquisitionState, DPC/API/PTACQST )
;
; Name:          PointingAcquisitionState
; Description:   Acquisition State Routine.
; On Entry:     EAX  N/A
;
; .....

```

```

; EBX      Frame Data Space
; ECX      N/A
; EDX      N/A
; EBP      Adapter Data Space
; ESI      N/A
; EDI      N/A

```

```

; Note:     Interrupts are in any state.

```

```

; On Return: EAX      Destroyed
;            EBX      Preserved
;            ECX      Destroyed
;            EDX      Destroyed
;            EBP      Preserved
;            ESI      Preserved
;            EDI      Preserved

```

```

; Flags:

```

```

; Note:     Interrupts preserved.

```

```

; Remarks:   This routine is called by DriverCallback.
;            It can be called at process or interrupt time.

```

```

; See Also:

```

```

; END_MANUAL_ENTRY

```

```

; .....
; public PointingAcquisitionState
; PointingAcquisitionState proc

```

```

; cmp      DebugMask, 0
; je       PointingAcquisitionStateNoMsg
; mov      eax, offset PointingAcqStateMsg
; cmp      eax, LastDebugMessage
; je       PointingAcquisitionStateNoMsg
; mov      mov      LastDebugMessage, eax
; push     eax
; push     DPCScreen
; call     OutputToScreen
; lea      esp, (esp + (2 * 4))
; PointingAcquisitionStateNoMsg:

```

```

; mov      edx, [ebp].IOStatusAddr
; in       al, dx
; test     al, SWEEPING_MASK
; je       PointingNotSweeping

; mov      esi, [ebp].IOTuningLowAddr
; mov      edi, [ebp].IOTuningHighAddr
; call     ReadWord
; shl     eax, 4
; mov      [ebp].Drift, eax

```

```

; mov      [ebp].DemodStatus, LOCKED

; xor      eax, eax
; mov      edx, [ebp].IOGateCountHighAddr
; out      dx, al

```

```

; mov      edx, [ebp].IOBtrControlAddr
; in       al, dx
; and      al, NOT (FREQ_PWR_MASK OR PHASE_PWR_MASK)
; out      dx, al

```

```

; in       al, dx
; or       al, 5
; out      dx, al

; mov      [ebp].NextStepCount, 1
; mov      [ebp].TiCount, 1000
; mov      [ebp].SearchLocFound, FALSE
; mov      [ebp].CurrentState, POINTING_TRACKING
; ret

```

```

; PointingNotSweeping:

```

```

; mov      eax, [ebp].MaxSsqf
; sub      eax, 2
; mov      [ebp].MaxSsqf, eax
; mov      edx, [ebp].IOctHaddr
; out      dx, al
; cmp      [ebp].TiCount, 0
; jne      PointingAcqExit

```

```

; mov      [ebp].CurrentState, INIT

```

```

; PointingAcqExit:
; ret

```

```

; PointingAcquisitionState      endp
; subttl -- PointingTrackingState --
; page

```

```

; .....
; BEGIN_MANUAL_ENTRY( PointingTrackingState, DPC/API/PTTRKST )

```

```

; Name:      PointingTrackingState
; Description: Acquisition State Routine.
; On Entry:  EAX      N/A
;            EBX      Frame Data Space
;            ECX      N/A
;            EDX      N/A
;            EBP      Adapter Data Space
;            ESI      N/A
;            EDI      N/A

```

```

; Note:      Interrupts are in any state.

```

```

; On Return: EAX      Destroyed
;            EBX      Preserved
;            ECX      Destroyed
;            EDX      Destroyed
;            EBP      Preserved
;            ESI      Preserved
;            EDI      Preserved

```

```

; Flags:

```

```

; Note:      Interrupts preserved.

```

```

; Remarks:   This routine is called by DriverCallback.
;            It can be called at process or interrupt time.

```

```

; See Also:

```

```

; END_MANUAL_ENTRY

```

```
public PointingTrackingState
PointingTrackingState proc
```

```
    cmp     DebugMask, 0
    je      PointingTrackingStateNoMsg
    mov     eax, offset PointingTrackingStateMsg
    cmp     eax, LastDebugMessage
    je      PointingTrackingStateNoMsg
    mov     eax, LastDebugMessage, eax
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    xor     eax, eax
    mov     edx, [ebp].IOReIsqfAddr
    in      al, dx
    mov     [ebp].SignalQuality, eax
```

```
PointingTrackingStateNoMsg:
    xor     eax, eax
    mov     edx, [ebp].IOReIsqfAddr
    in      al, dx
    mov     [ebp].SignalQuality, eax
```

```
    cmp     [ebp].TiCount, 0
    je      PointingTrackingExit
```

```
    mov     [ebp].TiCount, 1000
```

```
    mov     esi, [ebp].IOTuningLowAddr
    mov     edi, [ebp].IOTuningHighAddr
    call    ReadWord
```

```
    mov     ecx, [ebp].Drift
    add     ecx, OFFSET_THRESHOLD
    cmp     eax, ecx
    ja      PointingTrackingInit
```

```
    mov     ecx, [ebp].Drift
    sub     ecx, OFFSET_THRESHOLD
    cmp     eax, ecx
    jae     PointingTrackingExit
```

```
PointingTrackingInit:
    mov     [ebp].CurrentState, INIT
```

```
PointingTrackingExit:
    ret
```

```
PointingTrackingState endp
    subttl -- HaltState --
    page
```

```
; BEGIN_MANUAL_ENTRY( HaltState, DPC/API/HALTST )
```

```
; Name: HaltState
```

```
; Description: Acquisition State Routine.
```

```
; On Entry: EAX N/A
;           EBX Frame Data Space
;           ECX N/A
;           EDX N/A
;           EBP Adapter Data Space
;           ESI N/A
;           EDI N/A
```

```
; Note: Interrupts are in any state.
```

```
; On Return: EAX Destroyed
;            EBX Preserved
;            ECX Destroyed
;            EDX Destroyed
;            EBP Preserved
;            ESI Destroyed
;            EDI Preserved
```

```
; Note: Interrupts are in any state.
```

```
; On Return: EAX Destroyed
;            EBX Preserved
;            ECX Destroyed
;            EDX Destroyed
;            EBP Preserved
;            ESI Preserved
;            EDI Preserved
```

```
; Flags:
```

```
; Note: Interrupts preserved.
```

```
; Remarks: This routine is called by DriverCallback.
;           It can be called at process or interrupt time.
```

```
; See Also:
```

```
; END_MANUAL_ENTRY
```

```
public HaltState
HaltState proc
```

```
    cmp     DebugMask, 0
    je      HaltStateNoMsg
    mov     eax, offset HaltStateMsg
    cmp     eax, LastDebugMessage
    je      HaltStateNoMsg
    mov     eax, LastDebugMessage, eax
    push    eax
    push    DPCScreen
    call    OutputToScreen
    lea     esp, [esp + (2 * 4)]
    ret
```

```
HaltStateNoMsg:
```

```
HaltState endp
    subttl -- InitDemod --
    page
```

```
; BEGIN_MANUAL_ENTRY( InitDemod, DPC/API/INITMOD )
```

```
; Name: InitDemod
```

```
; Description: Acquisition State Routine.
```

```
; On Entry: EAX N/A
;           EBX Frame Data Space
;           ECX N/A
;           EDX N/A
;           EBP Adapter Data Space
;           ESI N/A
;           EDI N/A
```

```
; Note: Interrupts are in any state.
```

```
; On Return: EAX Destroyed
;            EBX Preserved
;            ECX Destroyed
;            EDX Destroyed
;            EBP Preserved
```

```
ESI    Preserved
EDI    Preserved
```

```
Flags:
```

```
Note:    Interrupts preserved.
```

```
Remarks: This routine is called by DriverCallback.
           It can be called at process or interrupt time.
```

```
See Also:
```

```
END_MANUAL_ENTRY
```

```
public InitDemod
proc
```

```
mov     [ebp].CurrentState, HALT
mov     [ebp].RxFreq, 0
mov     [ebp].ViterbiMode, 0
mov     [ebp].DemodCommand, HALT_MODE
mov     [ebp].SearchLoc, 1
mov     [ebp].Drift, 0
mov     [ebp].GLOffset, 0
mov     [ebp].TrackingMode, FALSE
```

```
;value = read_bits (STATUS_ADDR, TUNER_TYPE_MASK);
```

```
xor     eax, eax
mov     edx, [ebp].IOStatusAddr
in      al, dx
and     al, TUNER_TYPE_MASK
mov     cl, al
```

```
;value |= read_bits (UNIT_ID_ADDR, TUNER_TYPE_2_MASK);
```

```
mov     edx, [ebp].IOUnitIDAddr
in      al, dx
and     al, TUNER_TYPE_2_MASK
or      al, cl
```

```
cmp     al, SHARP
jne     InitDemodPanasonic
```

```
cmp     [ebp].DebugMask, 0
je      FillInTunerVars
push    eax
push    offset SharpTunerMsg
push    DPCScreen
call    OutputToScreen
lea     esp, [esp + (2 * 4)]
pop     eax
jmp     FillInTunerVars
```

```
InitDemodPanasonic:
```

```
cmp     al, PANASONIC
jne     InitDemodSharpCustom
```

```
cmp     [ebp].DebugMask, 0
je      FillInTunerVars
push    eax
push    offset PanasonicTunerMsg
push    DPCScreen
call    OutputToScreen
lea     esp, [esp + (2 * 4)]
```

```
pop     eax
jmp     FillInTunerVars
```

```
InitDemodSharpCustom:
```

```
cmp     al, SHARP_CUSTOM
jne     InitDemodExit

cmp     [ebp].DebugMask, 0
je      FillInTunerVars
push    eax
push    offset SharpCustomTunerMsg
push    DPCScreen
call    OutputToScreen
lea     esp, [esp + (2 * 4)]
pop     eax
```

```
FillInTunerVars:
```

```
mov     [ebp].TunerTypeFound, eax
mov     [ebp].ReacqGateCount, 0f0h
mov     [ebp].ReacqDeltaCount, 2
mov     [ebp].NomCountSearch, NOM_COUNT_REACQ - 75
mov     [ebp].SqfCheckPoints, 11
mov     [ebp].SqfCheckStepSize, 15
mov     [ebp].SqfDeltaCount, 8
```

```
InitDemodExit:
ret
```

```
InitDemod endp
```

```
subttl  -- ApplyDelay --
page
```

```
BEGIN_MANUAL_ENTRY( ApplyDelay, DPC/API/APPLYDEL )
```

```
Name:    ApplyDelay
```

```
Description: Acquisition State Routine.
```

```
On Entry:  EAX    N/A
            EBX    Frame Data Space
            ECX    N/A
            EDX    N/A
            EBP    Adapter Data Space
            ESI    N/A
            EDI    N/A
```

```
Note:    Interrupts are in any state.
```

```
On Return: EAX    Destroyed
            EBX    Preserved
            ECX    Destroyed
            EDX    Destroyed
            EBP    Preserved
            ESI    Preserved
            EDI    Preserved
```

```
Flags:
```

```
Note:    Interrupts preserved.
```

```
Remarks: This routine is called by DriverCallback.
           It can be called at process or interrupt time.
```

```
See Also:
```

; END_MANUAL_ENTRY

; public ApplyDelay
; proc

; Apply delay to T1 Counter

```
; cmp (ebp).T1Count, 0
; je ApplyDelayT2
; cmp (ebp).T1Count, eax
; jb ApplyDelayClearT1
; sub (ebp).T1Count, eax
; jmp ApplyDelayT2

ApplyDelayClearT1:
; mov (ebp).T1Count, 0
```

; Apply delay to T2 Counter

```
; ApplyDelayT2:
; cmp (ebp).T2Count, 0
; je ApplyDelayExit
; cmp (ebp).T2Count, eax
; jb ApplyDelayClearT2
; sub (ebp).T2Count, eax
; jmp ApplyDelayExit

ApplyDelayClearT2:
; mov (ebp).T2Count, 0
```

ApplyDelayExit:
ret

```
ApplyDelay endp
subttl -- CalculatorRxFreq --
page
```

; BEGIN_MANUAL_ENTRY(CalculatorRxFreq, DPC/API/CALCRXFQ)

; Name: CalculatorRxFreq

; Description: Acquisition State Routine.

| | | |
|-----------|-----|--------------------|
| On Entry: | EAX | N/A |
| | EBX | Frame Data Space |
| | ECX | N/A |
| | EDX | N/A |
| | EBP | Adapter Data Space |
| | ESI | N/A |
| | EDI | N/A |

; Note: Interrupts are in any state.

| | | |
|------------|-----|-----------|
| On Return: | EAX | Destroyed |
| | EBX | Preserved |
| | ECX | Destroyed |
| | EDX | Destroyed |
| | EBP | Preserved |
| | ESI | Preserved |
| | EDI | Preserved |

; Flags:

; Remarks:

; This routine is called by DriverCallback.
; It can be called at process or interrupt time.

; See Also:

; END_MANUAL_ENTRY

; Note: Interrupts preserved.

; public CalculatorRxFreq
; CalculateRxFreq proc

```
; USHORT_T freq, total, new_total, results;
sub esp, 2 * 4
```

```
; total = [esp + 0]
; results = [esp + 4]
; freq = esi
; new_total = edi
```

```
; freq = S.Rx_Freq - FREQ_BASE;
; freq += S.Gl_Offset;
; mov esi, [ebp].Rx_Freq
; sub esi, FREQ_BASE
; add esi, [ebp].Gl_Offset ; ESI = freq
```

```
; total = (freq * 2) / 729;
; mov eax, esi
; shl eax, 1
; xor edx, edx
; mov ecx, 729
; div ecx
; mov [esp + 0], eax ; EAX = EAX / 729
; total = EAX
```

```
; results = (total * 729) / 2;
; mov ecx, 729
; mul ecx
; shr eax, 1
; mov [esp + 4], eax ; EAX = total * 729
; results = eax
```

```
; freq = freq - results;
sub esi, eax
```

```
; new_total = total * 10;
; mov eax, [esp + 0]
; mov ecx, 10
; mul ecx
; mov edi, eax ; EAX = total * 10
; new_total = EAX
```

```
; total = (freq * 20) / 729;
; mov eax, esi
; mov ecx, 20
; mul ecx
; xor edx, edx
; mov ecx, 729
; div ecx
; mov [esp + 0], eax ; EAX = EAX / 729
; total = EAX
```

```
; results = (total * 729) / 20;
; mov ecx, 729
; mul ecx
; xor edx, edx
; mov ecx, 20
; div ecx ; EAX = total * 729
; EAX = EAX / 20
```

```

mov     [esp + 4], eax
;
; freq = freq - results;
sub     esi, eax
;
; new_total += total;
add     edi, [esp + 0]
;
; new_total *= 10;
mov     eax, edi
mov     ecx, 10
mul     ecx
mov     edi, eax
;
; total = (freq * 200) / 729;
mov     eax, esi
mov     ecx, 200
mul     ecx
xor     edx, edx
mov     ecx, 729
div     ecx
mov     [esp + 0], eax
;
; results = (total * 729) / 200;
mov     ecx, 729
mul     ecx
xor     edx, edx
mov     ecx, 200
div     ecx
mov     [esp + 4], eax
; results = EAX
;
; freq = freq - results;
sub     esi, eax
;
; new_total += total;
add     edi, [esp + 0]
;
; if (freq >= 2) new_total++;
cmp     esi, 2
jb      CalcGetChannelNumber
inc     edi
; new_total++
CalcGetChannelNumber:
;
; S.Channel_Number = SYNTH_FIRST_CHANNEL + new_total;
add     edi, SYNTH_FIRST_CHANNEL
mov     [ebp].ChannelNumber, edi
;
cmp     DebugMask, 0
je      NoChannelMsg
push    edi
push    offset ChannelNumberMsg
push    DPCScreen
call    OutputToScreen
lea     esp, [esp + (3 * 4)]
NoChannelMsg:
add     esp, 2 * 4
ret
;
CalculateRxFreq endp
subttl -- DriverCallback --
page
;
; .....
; BEGIN_MANUAL_ENTRY( DriverCallback, DPC/API/CALLBACK )

```

```

; results = EAX

```

```

; Name: DriverCallback

```

```

; Description: This routine will be executed once every second. It will
; detect if the hardware does not ack a transmission. If the
; hardware didn't ack then it will be reset, the transmission
; of that packet will be aborted and the next packet in the
; queue will be sent if there is one.

```

```

; On Entry:
; EAX N/A
; EBX @ Frame Data Space
; ECX N/A
; EDX N/A
; EBP @ Adapter Data Space
; ESI N/A
; EDI N/A

```

```

; Note: Interrupts are disabled.

```

```

; On Return:
; EAX Destroyed
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Destroyed
; EDI Destroyed

```

```

; Flags:

```

```

; Note: Interrupts disabled.

```

```

; Remarks: This routine is called by the MSM.
; After this call returns, the MSM will schedule another
; call back.
; It is called at interrupt time.

```

```

; See Also: MSM\MSMCallbackProcedure

```

```

; END_MANUAL_ENTRY

```

```

; .....

```

```

public DriverCallback
align 16
DriverCallback proc

```

```

    cmp [ebp].TunerTypeFound, INVALID_TUNER
    jne DemodInitialized
    call InitDemod

```

```

; Check Rx Frequency

```

```

; DemodInitialized:

```

```

; cmp [ebp].RxFreq, 1330 * 10
; je CallBackCheckState
; mov [ebp].RxFreq, 1330 * 10

```

```

; cmp [ebp].RxFreq, 0
; jne CallBackCheckState
; mov eax, GlobalRxFreq

```

```

; RxFreq = GlobalRxFreq * 10

```

```

; mov ecx, 10

```

```

; mul ecx

```

```

; mov [ebp].RxFreq, eax

```

```

    cmp     [ebp].TrackingMode, FALSE
    je      CheckRxFreqSetMode
    call    CalculateRxFreq
    mov     [ebp].DemodCommand, ACQUIRE_MODE

; Possibly set the CurrentState depending on DemodCommand
;
CallBackCheckState:
    cmp     [ebp].DemodCommand, ACQUIRE_MODE
    je      CallBackSetInit
    cmp     [ebp].DemodCommand, POINTING_MODE
    jne     CallBackCheckHalt
CallBackSetInit:
    mov     [ebp].CurrentState, INIT
    call    CallBackApplyDelay
    jmp     CallBackWatchDog
CallBackCheckHalt:
    cmp     [ebp].DemodCommand, HALT_MODE
    jne     CallBackApplyDelay
    mov     [ebp].CurrentState, HALT

; Apply delay
;
CallBackApplyDelay:
    mov     eax, 15
    call    ApplyDelay

    mov     eax, [ebp].CurrentState
    mov     esi, StateTbl[eax * 4]
    call    esi

CallBackWatchDog:
    ret

    mov     eax, [ebp].BufferCount
    cmp     eax, [ebp].WatchBufferCount
    mov     [ebp].WatchBufferCount, eax
    jne     CallBackExit
    call    RefreshMipsStats

rFrames
    mov     eax, [ebp].MipsZeroAddrFrames

    cmp     eax, [ebp].WatchOldRejected
    mov     [ebp].WatchOldRejected, eax
    je      CallBackExit

    call    DriverISR
    mov     edx, [ebp].PicAddress
    in      al, dx
    SLOW
    or      eax, [ebp].PicMask
    out     dx, al
    SLOW
    in      al, dx
    SLOW
    and     eax, [ebp].PicUnMask
    out     dx, al

CallBackExit:
    ret

```

```

DriverCallBack endp
    public DriverSend
    subttl -- DriverSend --
    page
; .....
; BEGIN_MANUAL_ENTRY( DriverSend, DPC/API/SEND )
;
; Name: DriverSend
; Description: This routine will transfer the packet described in the
;              TCB to the NIC and initiate the send. TxStartTime and
;              RetryCounter must be set to enable the deadline timer.
;
; On Entry:    EAX N/A
;              EBX @ Frame Data Space
;              ECX Padded Packet Length
;              EDX N/A
;              EBP @ Adapter Data Space
;              ESI @ TCB
;              EDI N/A
;
; Note: Interrupts are disabled.
;
; On Return:   EAX Destroyed
;              EBX Preserved
;              ECX Destroyed
;              EDX Destroyed
;              EBP Preserved
;              ESI Destroyed
;              EDI Destroyed
;
; Flags:
;
; Note: Interrupts disabled.
;
; Remarks: This routine is called by the MSM media module.
;           It is called at process or interrupt time.
;
; See Also: ETHERTSM\EtherTSM\DriverSend
;            ETHERTSM\MediaSendRaw8023
;            ETHERTSM\MediaSendEthernetII
;            ETHERTSM\MediaSend8022Over8023
;            ETHERTSM\MediaSend8022Snap
;
; END_MANUAL_ENTRY
; .....
;
; align 16
; DriverSend proc
;
;     lea     edi, [esi].TCBMediaHeader
;     cmp     word ptr [edi+12], 0608h
;     je      DriverSendArp
;
;     cmp     [ebp].AgentSendRoutine, 0
;     je      DriverSendExit
;
;     t back if we can't
;
;     push    ecx
;     padded size
;     push    esi
;
;     Address of TCB
;     call    [ebp].AgentSendRoutine ; Give it to Slip Handler
;
; Give it

```

```

pop     esi
pop     ecx
ret

DriverSendExit:
inc     [ebp].MSMTxFreeCount
jmp     EtherTSMFastSendComplete

DriverSendArp:
; We're going to assume that the entire request is in the first
; fragment. Verify it first.
;
mov     edi, [esi].TCBFragStructPtr
cmp     dword ptr [edi+0], 1
jne     DriverSendExit
cmp     dword ptr [edi+8], 28
jb      DriverSendExit
; Make sure sender and target ip addr are different
;
mov     edi, [edi+4]
; EDI -> ARP request
mov     eax, [edi+28-14]
; EAX = senders IP
cmp     eax, [edi+38-14]
; Same as target IP?
je      DriverSendExit
; Jump out if it is
;
push    esi
push    edi
; Save send ECB
; Save ARP offset
;
mov     esi, 1514
call    MSNAllocaterCB
; Get an ECB
pop     edi
or      eax, eax
jne     DriverSendReturnARP
; ESI -> reply ECB
; EDI -> request data
;
lea     eax, [esi+RPacketEnvelope]
mov     [esi].RPacketOffset, eax
; EAX -> beginning
mov     [esi].RPacketSize, 60
; Store into ECB
mov     [esi].RPacketLength, 60
; ARP reply size
; (ethernet min size)
push    esi
mov     esi, eax
; Save reply ECB
; ESI -> reply data
;
; ESI -> reply data
; EDI -> request data
;
; Set reply->dest_addr to our node address
;
mov     eax, dword ptr [ebx].MLIDNodeAddress+0
mov     dword ptr [esi+0], eax
mov     ax, word ptr [ebx].MLIDNodeAddress+4
mov     word ptr [esi+4], ax
; Set reply->source_addr to (0x06 0x06 0x06 0x06 0x06 0x06)
;
mov     word ptr [esi+6], 0606h
mov     dword ptr [esi+8], 06060606h
; Set reply->type to (0x08 0x06)

```

```

mov     word ptr [esi+12], 0608h
; Set reply hardware type(0x00 0x01), protocol type(0x08 0x00)
;
mov     dword ptr [esi+14], 00080100h
; Set reply hardware size(0x06), protocol size(0x04)
; and operation(0x00 0x02 for ARP reply)
;
mov     dword ptr [esi+18], 02000406h
; Set reply senders ethernet addr to (0x06 0x06 0x06 0x06 0x06 0x06)
;
mov     dword ptr [esi+22], 06060606h
mov     word ptr [esi+26], 0606h
; Set reply senders ip addr to the request target ip addr
;
mov     eax, [edi+38-14]
mov     [esi+28], eax
; request->target_ip
; Set reply target ethernet addr to our node addr
;
mov     eax, dword ptr [ebx].MLIDNodeAddress+0
mov     dword ptr [esi+32], eax
mov     ax, word ptr [ebx].MLIDNodeAddress+4
mov     word ptr [esi+36], ax
; Set reply target ip addr to request senders ip addr
;
mov     eax, dword ptr [edi+28-14]
mov     dword ptr [esi+38], eax
; request->senders_ip
;
pop     esi
mov     edi, 1514
xor     eax, eax
mov     ecx, [esi].RPacketSize
push    ebp
call    EtherTSMFastProcessGetRCB
pop     ebp
jne     DriverSendReturnARP
MSMReturnRCB
DriverSendReturnARP:
pop     esi
inc     [ebp].MSMTxFreeCount
; Restore send ECB
; Add a send resource
; EtherTSMFastSendComplete
jmp

DriverSend
endp

extrn   DoEndOfInterrupt: near
extrn   SetHardwareInterrupt: near
extrn   ClearHardwareInterrupt: near
TestDriverISR
proc
ebp, OurAdapterDataSpace
mov     ebx, [ebp].MSMDefaultVirtualBoard
movzx   ecx, [ebx].MLIDInterrupt
call    DoEndOfInterrupt
inc     [ebp].GotInterrupt
xor     eax, eax
ret
TestDriverISR
endp

```

```
subttl  -- DriverISR --
page
```

```
; BEGIN_MANUAL_ENTRY( DriverISR, DPC/API/ISR )
```

```
; Name: DriverISR
```

```
; Description: This routine handles packet reception.
```

```
; On Entry: EAX N/A
```

```
; EBX N/A
```

```
; ECX N/A
```

```
; EDX N/A
```

```
; EBP @ Adapter Data Space
```

```
; ESI N/A
```

```
; EDI N/A
```

```
; Note: Interrupts are disabled.
```

```
; On Return: EAX Destroyed
```

```
; EBX Destroyed
```

```
; ECX Destroyed
```

```
; EDX Destroyed
```

```
; EBP Destroyed
```

```
; ESI Destroyed
```

```
; EDI Destroyed
```

```
; Flags:
```

```
; Note: Interrupts disabled.
```

```
; Remarks: This routine is called by the MSM.
; It is called at interrupt time.
```

```
; See Also: MSM\MSMInterruptProcedure
```

```
; END_MANUAL_ENTRY
```

```
align 16
public DriverISR
proc
```

```
; /* Set the adapters ram ptr to the next rbd to receive from */
; output(bid_base_addr + MSG_RAM_PTR, rbd_base_addr + 2*curr_adap_rbd);
```

```
DebugMessage DEBUG_ISR_ALL, ISREnterMsg
mov edx, [ebp].IOMsgRamPtr
mov eax, [ebp].CurrentAdapterRBD
shl eax, 1
add eax, RBD_BASE_ADDR
out dx, ax
; Set Adapter Ram Ptr
```

```
; /* Keep processing packets until no more are left */
```

```
; /*
```

```
; * NOTE: We are assuming that anyone looping back to DriverISRLoop
```

```
; * has set the MSR_RAM_PTR to the next RBD to examine.
```

```
; */
```

```
; while ((status = inport (bid_base_addr + MSG_RAM)) & EMPTY)
```

```
DriverISRLoop:
```

```
xor eax, eax
; Clear upper status
```

```
mov edx, [ebp].IOMsgRam
in ax, dx
mov [ebp].IntStatus, eax

test eax, EMPTY
je DriverISRExit

inc [ebp].BufferCount
DebugMessage1 DEBUG_ISR, DebugRBDReceived, [ebp].CurrentAdapterRBD
```

```
; /* Jump if this is an error packet */
; if (status & 0x8F)
```

```
test [ebp].IntStatus, STATUS_ERROR
jne DriverISRBadPacket
; Any error bits set?
; Jump if not
```

```
; /* Heres a good packet. See if we have a buffer for it. */
; if (!global_pool[curr_rbd].buf_ptr)
```

```
mov esi, [ebp].CurrentECB
or esi, esi
jne DriverISRAddToECB
; Is this more of
; the last packet?
; Jump if it is.
```

```
if TIMESTAMP
; mov al, 'r'
; push eax
; call DPCTimestamp
; lea esp, [esp + 4]
endif
```

```
mov esi, 1514
call MSMAllocateECB
or eax, eax
jne DriverISRNoECB
; Max ECB size.
; Get an ECB
; Jump if no ECB.
```

```
; !!!! Satellite header is 12 bytes, EII is 14 bytes.
```

```
; !!!! Add 2 to offset to prevent double copy of turbo internet packets.
```

```
lea edi, [esi+RPacketEnvelope+2] ; EDI -> beginning
mov [esi].RPacketOffset, edi ; Store into ECB
mov [esi].RPacketSize, 0 ; Clear size
mov [ebp].CurrentECB, esi ; Store if split packet
jmp -short DriverISRReadSize
```

```
DriverISRAddToECB:
mov edi, [esi].RPacketOffset
add edi, [esi].RPacketSize
```

```
DriverISRReadSize:
```

```
; /* ESI(curr_rbd) will be used a lot. Let's try to keep it intact. */
; /* Retrieve the length of the packet */
; length = inport(bid_base_addr + MSG_RAM);
```

```
xor eax, eax
mov edx, [ebp].IOMsgRam
in ax, dx
; Clear upper word
; Msg Ram I/O port
; Get size of packet
```

```
add [esi].RPacketSize, eax
DebugMessage1 DEBUG_ISR, DebugRBDSize, eax
; Add to ECB size
```

```
; word_length = (length & 3) ? (length / 4) + 1 : (length/4);
```



```

; /* ECX = length = total length of all buffers */
; p_length = (global_pool[first_buffer_rbd].buf_ptr) + 3;
; if (((*p_length + 12) > length) || ((*p_length + 12) < (length - 16)))
; {
;     mov     ecx, [esi].RPacketSize
;     mov     edx, [esi].RPacketOffset
;     and     edx, [edx+6]
;     add     edx, 0ffffh
;     cmp     edx, 12
;     jne     edx, ecx
;     ja      DriverISRFilterSkipRBDsLen
;     sub     ecx, 16
;     cmp     edx, ecx
;     jb      DriverISRFilterSkipRBDsLen
;     filter[i].total_count++;
;     inc     (edi).FilterTotalCount
;     ; Bump filter total count
; }
; /* Check address seq numbers and update stats
; p_seq_num = (global_pool[first_buffer_rbd].buf_ptr) + 2;
;     mov     edx, [esi].RPacketOffset
;     mov     eax, [edx+8]
;     if (*p_seq_num && filter[i].seq_num && (*p_seq_num != filter[i].seq_num
;     {
;         filter[i].seq_count++;
;         filter[i].seq_num = *p_seq_num + 1;
;     }
;     else if (!*p_seq_num)
;         filter[i].seq_num = 1;
;     else if (!filter[i].seq_num)
;         filter[i].seq = *p_seq_num + 1;
;     or     eax, eax
;     jne     DriverISRFilterNoPacketSeq
;     cmp     (edi).FilterSeqNum, 0
;     jne     DriverISRFilterNoFilterSeq
;     cmp     eax, [edi].FilterSeqNum
;     jne     DriverISRFilterSeqNoMatch
;     inc     (edi).FilterSeqNum
; }
; /* Discard frames if it's a "stats only" channel */
; if (rx_cntl[cc].flags & BICDD_FLAGS_STATS_ONLY)
; {
;     DriverISRFilterCallESR:
;     mov     eax, [ebx].RxEsr
;     or     eax, eax
;     je      DriverISRDidnWantECB
;     cmp     eax, 0fffffh
;     jne     DriverISRNotOurs
; }
; public DriverISRInternet
; DriverISRInternet:
;     INT_3
;     lea     edi, [esi+RPacketEnvelope]
;     mov     ebx, [ebp].MSMDefaultVirtualBoard
; }
;
;     mov     ecx, [esi].RPacketOffset
;     ; Make sure its not a data feed address. We don't know what to do
;     ; with these yet.
;     mov     al, [ecx+0]
;     mov     ah, [ecx+2]
;     and     al, 3
;     cmp     al, 3
;     je      MightBeDataStreamPacket
;     and     al, 1
;     cmp     al, 1
;     jne     NotDataStreamPacket
;     MightBeDataStreamPacket:
;     cmp     ah, 0fffh
;     jne     NotDataStreamPacket
;     ; Data stream, go into debugger.
;     ;
;     ;
;     INT_3
;     MSMReturnRCB
;     jmp     DriverISRLoop
; }
; Break into debugger
; Return it
; Get next packet
;
; Fill in EII Destination with our node address
;
;     mov     eax, dword ptr [ebx].MLIDNodeAddress
;     mov     [edi+0], eax
;     mov     ax, word ptr [ebx].MLIDNodeAddress+4
;     mov     [edi+4], ax
;     ; Fill in EII Source Address with (0x0a 0x0a 0x0a 0x0a 0x0a 0x0a)
;     ;
;     mov     word ptr [edi+6], 0a0ah
;     mov     dword ptr [edi+8], 0a0a0a0ah
;     mov     word ptr [edi+6], 0606h
;     mov     dword ptr [edi+8], 06060606h
;     ; Fill in EII type field with IP type (0x08 0x00)
;     ;
;     mov     word ptr [edi+12], 0008
;     ; Force IP PID(800h hf/lo) since
;     ; DPC is unaware of it
;     lea     ecx, [esi + RFragmentCount]
;     push    ecx
;     call    [DPCRxFrame]
;     add     esp, 4
;     movzx   ecx, word ptr [esi + RPacketEnvelope + 14 + 2]
;     xchg    ch, cl
;     add     ecx, 14
;     cmp     ecx, 3ch
;     jae     RPacketIsPadded
;     mov     ecx, 3ch
;     RPacketIsPadded:
;     ; vvv DEBUG
;     mov     eax, ecx
;     cmp     eax, 400
;     jbe     DebugRExit
;     cmp     eax, [ebp].LargestRx
;     jbe     DebugRXave
; }

```

```

mov     [ebp].LargestRx, eax

DebugRxAv:
inc     eax, [ebp].TotalLargerX
add     edi, [ebp].NumberLargerX
mov     [ebp].TotalLargerX, eax
xor     edx, edx
div     edi
mov     [ebp].AveLargerX, eax

DebugRxExit:
; *** DEBUG
;
; mov     edi, 1514
; ecx     ; Max Packet size
; push    al, 'R'
; push    eax
; call    DPCTimestamp
; lea     esp, [esp + 4]
; pop     ecx
; endif

xor     eax, eax
; Good packet

push    ebp
call    EthernetFastProcessGetRCB
pop     ebp
jne     DriverISRLoop
; no ecb returned
jmp     DriverISRDidntWantECB
hese

DriverISRNotOurs:
push    esi
call    eax
pop     esi
or     eax, eax
je     DriverISRLoop

DriverISRDidntWantECB:
MSMReturnRCB
jmp     DriverISRLoop

DriverISRFilterNext:
add     edi, size FilterStruct
lea     eax, [ebp].Filter(MAX_ADDR * size FilterStruct)
cmp     edi, eax
jbe     DriverISRFilterLoop
; Couldn't find filter address. Clean up.
;

MSMReturnRCB
DebugMessage6  DEBUG_ISR_ALL, FilterNone, (edx+0), (edx+1), (edx+2), (e
dx+3), (edx+4), (edx+5)
jmp     DriverISRLoop

DriverISRFilterSeqNoMatch:
inc     eax
inc     [edi].FilterSeqCount
mov     [edi].FilterSeqNum, eax
jmp     DriverISRFilterCallesR

DriverISRFilterNoFilterSeq:
inc     eax
mov     [edi].FilterSeqNum, eax
jmp     DriverISRFilterCallesR

DriverISRFilterNoPacketSeq:
mov     [edi].FilterSeqNum, 1
jmp     DriverISRFilterCallesR

DriverISRFilterSkipRBDSLen:
MSMReturnRCB
DebugMessage2  DEBUG_ISR_ALL, FilterRBDLen, ecx, edx
jmp     DriverISRLoop

DriverISRNoECB:
DebugMessage  DEBUG_ISR, NoECBMsg
jmp     DriverISRBadNextRBD

DriverISRBadPacket:
mov     esi, [ebp].IntStatus
test    esi, FRAMING_ERR
je     DriverISRCheckAbort
DebugMessage  DEBUG_ISR, FramingErrMsg

DriverISRCheckAbort:
test    esi, ABORT
je     DriverISRCheckAlign
DebugMessage  DEBUG_ISR, AbortMsg

DriverISRCheckAlign:
test    esi, ALIGN_ERR
je     DriverISRCheckOverrun
DebugMessage  DEBUG_ISR, AlignErrMsg

DriverISRCheckOverrun:
test    esi, OVERRUN_ERR
je     DriverISRCheckDES
DebugMessage  DEBUG_ISR, OverrunErrMsg

DriverISRCheckDES:
test    esi, DES_ERR
je     DriverISRCheckCRC
DebugMessage  DEBUG_ISR, DESErrMsg

DriverISRCheckCRC:
test    esi, CRC_ERR
je     DriverISRErrorStats
DebugMessage  DEBUG_ISR, CRCErrMsg

DriverISRErrorStats:
DebugMessage  DEBUG_ISR, ReturnMsg

DriverISRBadNextRBD:
mov     edx, [ebp].IOMsgRamPtr
mov     eax, [ebp].CurrentAdapterRBD
shl     eax, 1
add     ecx, RBD_BASE_ADDR
out     dx, ax

```

```
mov     cdx, [ebp].IOMsgRam
xor     eax, eax
out     dx, ax

mov     eax, RBD_BUFFER_SIZE
out     dx, ax

mov     eax, [ebp].CurrentAdapterRBD
inc     eax
cmp     eax, ADAP_RBD_NUM
jb      DriverISRBadRBDWrap
xor     eax, eax
DriverISRBadRBDWrap:
mov     [ebp].CurrentAdapterRBD, eax

DebugMessage1 DEBUG_ISR_ALL, AdapterRBDMsg, eax
mov     edx, [ebp].IOMsgRamPtr
shl     eax, 1
add     eax, RBD_BASE_ADDR
out     dx, ax
jmp     DriverISRLoop
```

```
DriverISRExit:
mov     edx, [ebp].IOMsgRamPtr
mov     eax, 0c3a0h
out     dx, ax

mov     eax, [ebp].CurrentAdapterRBD
mov     edx, [ebp].IOMsgRam
out     dx, ax

DebugMessage1 DEBUG_ISR_ALL, ISRExitMsg, eax

mov     edx, [ebp].IOStatus
in      ax, dx
ret
```

```
DriverISR      endp
subttl  -- DriverDisableInterrupt --
page
; .....
; BEGIN_MANUAL_ENTRY( DriverDisableInterrupt, DPC/API/DISINT )
;
; Name:      DriverDisableInterrupt
;
; Description: This routine will disable the adapters ability to
;              interrupt the host.
;
; On Entry:  EAX  N/A
;            EBX  N/A
;            ECX  N/A
;            EDX  N/A
;            EBP  @ Adapter Data Space
;            ESI  N/A
;            EDI  N/A
;
; Note:      Interrupts are disabled.
;
; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Preserved
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
;
; Flags:
;
; Note:      Interrupts disabled.
;
; Remarks:   This routine is called by the MSM.
;
; See Also:  DriverDisableInterrupt
;
; END_MANUAL_ENTRY
; .....
; align 16
```

```
; EBP  Preserved
; ESI  Preserved
; EDI  Preserved
;
; Flags:
;
; Note:      Interrupts disabled.
;
; Remarks:   This routine is called by the MSM.
;
; See Also:  DriverEnableInterrupt
;
; END_MANUAL_ENTRY
; .....
; align 16
DriverDisableInterrupt proc
xor     eax, eax
ret
DriverDisableInterrupt endp
subttl  -- DriverEnableInterrupt --
page
; .....
; BEGIN_MANUAL_ENTRY( DriverEnableInterrupt, DPC/API/ENINT )
;
; Name:      DriverEnableInterrupt
;
; Description: This routine will enable the adapters ability to
;              interrupt the host.
;
; On Entry:  EAX  N/A
;            EBX  N/A
;            ECX  N/A
;            EDX  N/A
;            EBP  @ Adapter Data Space
;            ESI  N/A
;            EDI  N/A
;
; Note:      Interrupts are disabled.
;
; On Return: EAX  Destroyed
;            EBX  Preserved
;            ECX  Preserved
;            EDX  Destroyed
;            EBP  Preserved
;            ESI  Preserved
;            EDI  Preserved
;
; Flags:
;
; Note:      Interrupts disabled.
;
; Remarks:   This routine is called by the MSM.
;
; See Also:  DriverDisableInterrupt
;
; END_MANUAL_ENTRY
; .....
; align 16
```

```
DriverEnableInterrupt proc
```

```
ret
```

```
DriverEnableInterrupt endp
```

```
public DriverReset
```

```
subttl -- DriverReset --
```

```
page
```

```

; .....
; BEGIN_MANUAL_ENTRY( DriverReset, DPC/API/RESET )
;
; Name: DriverReset
; Description: This routine will reset and initialize the NIC.
; On Entry: EAX N/A
; EBX 0 Frame Data Space
; ECX N/A
; EDX N/A
; EBP 0 Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are disabled.
; On Return: EAX 0 if successful (otherwise points to error message)
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Destroyed
; EDI Destroyed
; Flags:
; Note: Interrupts disabled.
; Remarks: This routine is called by the MSM media module.
; It is called at process time.
; See Also: ETHERTSM\EtherTSMReset
; END_MANUAL_ENTRY
; .....

```

```
DriverReset proc near
```

```
inc (ebp).AdapterResetCount ; Increment stat counter.
```

```
xor eax, eax
```

```
ret
```

```
DriverReset endp
```

```
DefaultRxFrame proc
```

```
ret
```

```
DefaultRxFrame endp
```

```
extrn LSLGetStackIDFromName: near
```

```
ProtocolBindEvent proc
```

```

; .....
;
; On Entry: EAX N/A
; EBX 0 Frame Data Space
; ECX N/A
; EDX N/A
; EBP 0 Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are disabled.
; On Return: EAX 0 if successful (otherwise points to error message)
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Destroyed
; EDI Destroyed
; Flags:
; Note: Interrupts disabled.
; Remarks: This routine is called by the MSM media module.
; It is called at process time.
; See Also: ETHERTSM\EtherTSMReset
; END_MANUAL_ENTRY
; .....

```

```
ProtocolBindEvent proc near
```

```
inc (ebp).AdapterResetCount ; Increment stat counter.
```

```
xor eax, eax
```

```
ret
```

```
ProtocolBindEvent endp
```

```
DefaultTxFrame proc
```

```
ret
```

```
DefaultTxFrame endp
```

```
extrn LSLGetStackIDFromName: near
```

```
ProtocolUnbindEvent proc
```

```

; .....
;
; On Entry: EAX N/A
; EBX 0 Frame Data Space
; ECX N/A
; EDX N/A
; EBP 0 Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are disabled.
; On Return: EAX 0 if successful (otherwise points to error message)
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Destroyed
; EDI Destroyed
; Flags:
; Note: Interrupts disabled.
; Remarks: This routine is called by the MSM media module.
; It is called at process time.
; See Also: ETHERTSM\EtherTSMReset
; END_MANUAL_ENTRY
; .....

```

```
ProtocolUnbindEvent proc near
```

```
inc (ebp).AdapterResetCount ; Increment stat counter.
```

```
xor eax, eax
```

```
ret
```

```
ProtocolUnbindEvent endp
```

```
DefaultTxFrame proc
```

```
ret
```

```
DefaultTxFrame endp
```

```
extrn LSLGetStackIDFromName: near
```

```
ProtocolUnbindEvent proc
```

```

; .....
;
; On Entry: EAX N/A
; EBX 0 Frame Data Space
; ECX N/A
; EDX N/A
; EBP 0 Adapter Data Space
; ESI N/A
; EDI N/A
; Note: Interrupts are disabled.
; On Return: EAX 0 if successful (otherwise points to error message)
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Destroyed
; EDI Destroyed
; Flags:
; Note: Interrupts are disabled.
; Remarks: This routine is called by the MSM media module.
; It is called at process time.
; See Also: ETHERTSM\EtherTSMReset
; END_MANUAL_ENTRY
; .....

```

```
ProtocolUnbindEvent proc near
```

```
inc (ebp).AdapterResetCount ; Increment stat counter.
```

```
xor eax, eax
```

```
ret
```

```
lea edx, IPName
```

```
call LSLGetStackIDFromName ; Return Stack ID in EBX
```

```
or eax, eax
```

```
jne short ProtocolBindExit
```

```
mov esi, [esp + Parm0]
```

```
cmp [esi+4], ebx ; IP Stack?
```

```
jne short ProtocolBindExit ; Nope
```

```
mov edx, [esi]
```

```
mov ebp, OurAdapterDataSpace ; EDX = Bound board number
```

```
xor ecx, ecx
```

```
ProtocolBindLoop:
```

```
mov ebx, [ebp+MSMVirtualBoardLink][ecx*4]
```

```
or ebx, ebx
```

```
jz ProtocolBindNext
```

```
cmp [ebx].MLIDBoardNumber, dx
```

```
jne ProtocolBindNext
```

```
mov eax, 1514
```

```
mov [ebx].MLIDMaximumSize, eax
```

```
sub eax, 14
```

```
mov [ebx].MLIDMaxRecvSize, eax
```

```
mov [ebx].MLIDRecvSize, eax
```

```
ProtocolBindNext:
```

```
inc ecx
```

```
cmp ecx, 4
```

```
jb ProtocolBindLoop
```

```
ProtocolBindExit:
```

```
CPop
```

```
ret
```

```
ProtocolBindEvent endp
```

```
ProtocolUnbindEvent proc
```

```
CPush
```

```
lea edx, IPName
```

```
call LSLGetStackIDFromName ; Return Stack ID in EBX
```

```
or eax, eax
```

```
jne short ProtocolUnbindExit
```

```
mov esi, [esp + Parm0]
```

```
cmp [esi+4], ebx ; IP Stack?
```

```
jne short ProtocolBindExit ; Nope
```

```
mov edx, [esi]
```

```
mov ebp, OurAdapterDataSpace ; EDX = Bound Board Number
```

```
xor ecx, ecx
```

```
ProtocolUnbindLoop:
```

```
mov ebx, [ebp+MSMVirtualBoardLink][ecx*4]
```

```
or ebx, ebx
```

```
jz ProtocolUnbindNext
```

```
cmp [ebx].MLIDBoardNumber, dx
```

```
jne ProtocolUnbindNext
```

```
mov eax, 1494
```

```
mov [ebx].MLIDMaximumSize, eax
```

```
sub eax, 14
```

```
mov [ebx].MLIDMaxRecvSize, eax
```

```
mov [ebx].MLIDRecvSize, eax
```

```
ProtocolUnbindNext:
```

```
inc ecx
cmp ecx, 4
jb ProtocolUnbindLoop
CPop
ret
```

```
ProtocolUnbindEvent endp
```

```
subttl -- DriverInit --
page
```

```
; BEGIN_MANUAL_ENTRY( DriverInit, DPC/API/INIT )
```

```
; Name: DriverInit
```

```
; Description: This routine will call EthernSMRegisterHSM,
MSMParseDriverParameters, MSMRegisterHardwareOptions,
MSMSetHardwareInterrupt, MSMRegisterMLID, initialize
variables in the Adapter Data Space and reset/initialize
the card.
```

```
; On Entry: EAX N/A
; EBX N/A
; ECX N/A
; EDX N/A
; EBP N/A
; ESI N/A
; EDI N/A
```

```
; Note: Interrupts are enabled.
```

```
; On Return: EAX 0 if successful (otherwise it points to error message)
; EBX Preserved
; ECX Destroyed
; EDX Destroyed
; EBP Preserved
; ESI Preserved
; EDI Preserved
```

```
; Flags:
```

```
; Note: Interrupts preserved.
```

```
; Remarks: This routine is called by the OS at load time.
; It is called at process time.
```

```
; See Also: MSM\MSMParseDriverParameters
MSM\MSMRegisterHardwareOptions
MSM\MSMSetHardwareInterrupts
MSM\MSMRegisterMLID
MSM\MSMScheduleInterruptCallBack
MSM\MSMScheduleAESCallBack
MSM\MSMEnablePolling
DriverReset
```

```
END_MANUAL_ENTRY
```

```
.....
```

```
DriverInit proc
```

```
CPush
if TIMESTAMP
```

```
lea eax, DPCTB
mov timestamp_begin, eax
mov timestamp_index, eax
add eax, TIMESTAMP_BUFFER_SIZE
mov timestamp_end, eax
```

```
endif
```

```
; .....
```

```
; Fill in Driver Parameter Block fields.
```

```
; .....
```

```
; mov DriverStackPointer, esp ; Fill in stack ->.
```

```
lea esi, DriverParameterBlock ; ESI -> Parm block.
call EthernSMRegisterHSM ; Get EBX.
jnz DriverInitError ; Jump if error.
```

```
; Yuck! We'll have to adjust the receive size down, since
; Hughes can't handle full 1500 byte packets with tunneling.
```

```
; mov [ebx].MLIDMaximumSize, 1494
```

```
; .....
```

```
; EBX -> Frame Data Space(Config Table).
```

```
; Let MSM Parse the command line.
```

```
; .....
```

```
; mov GlobalRxFreq, DEFAULT_RX_FREQ
```

```
; mov eax, NeedsIOPort0Bit OR NeedsInterrupt0Bit OR CAN_SET_NODE_ADDRE
```

```
SS
```

```
lea ecx, AdapterOptions
```

```
call MSMParseDriverParameters
```

```
jnz DriverInitError ; Jump if error.
```

```
; .....
```

```
; Let MSM Register the hardware options.
```

```
; .....
```

```
; call MSMRegisterHardwareOptions
```

```
cmp eax, 1
```

```
ja DriverInitError
```

```
je DriverInitExit
```

```
mov OurAdapterDataSpace, ebp
```

```
mov DPCRXFrame, offset DefaultRXFrame
```

```
; Get a timer resource tag so that we can delay ourselves.
```

```
; push TimerSignature
```

```
; push offset TimerDesc
```

```
; push DriverModuleHandle
```

```
; call AllocateResourceTag
```

```
extrn RegisterForEventNotification: near
extrn UnRegisterEventNotification: near
```

```
lea esp, (esp + (3 * 4))
mov (ebp).TimerTag, eax
or eax, eax
lea eax, ErrorAllocatingRtTagMessage
je DriverInitErrorReturn
```

```
; Get a resource tag for interrupts.
```

```
push InterruptSignature
push offset InterruptRtTagMessage
push DriverModuleHandle
call AllocateResourceTag
add esp, (3 * 4)
mov (ebp).ISRTag, eax
or eax, eax
lea eax, ErrorAllocatingRtTagMessage
je DriverInitErrorReturn
```

```
; Get a resource tag for event notification.
```

```
push EventSignature
push offset EventRtTagMessage
push DriverModuleHandle
call AllocateResourceTag
add esp, (3 * 4)
mov (ebp).EventTag, eax
or eax, eax
lea eax, ErrorAllocatingRtTagMessage
je DriverInitErrorReturn
```

```
; Register for protocol bind event.
```

```
mov eax, (ebp).EventTag
push offset ProtocolBindEvent
push 0
push EVENT_PRIORITY_OS
push EVENT_PROTOCOL_BIND
push eax
call RegisterForEventNotification
add esp, (4 * 5)
mov (ebp).ProtocolBindID, eax
or eax, eax
je DriverInitNoBindEvent
```

```
mov eax, (ebp).EventTag
push offset ProtocolUnbindEvent
push 0
push EVENT_PRIORITY_OS
push EVENT_PROTOCOL_UNBIND
push eax
call RegisterForEventNotification
add esp, (4 * 5)
mov (ebp).ProtocolUnbindID, eax
or eax, eax
jne DriverInitSetPorts
```

```
DriverInitNoBindEvent:
```

```
mov eax, 1494
```

```
mov (ebx).MLIDMaximumSize, eax
sub eax, 14
mov (ebx).MLIDMaxRecvSize, eax
mov (ebx).MLIDRecvSize, eax
```

```
DriverInitSetPorts:
```

```
.....\
```

```
; Set and check the adapters base I/O.
```

```
.....
```

```
movzx ecx, (ebx).MLIDIOPortsAndLengths
mov (ebp).IORxData, ecx
```

```
add ecx, 2
mov (ebp).IOAutoInc, ecx
add ecx, 2
mov (ebp).IOStatus, ecx
add ecx, 2
mov (ebp).IOControl, ecx
```

```
add ecx, 2
mov (ebp).IOMsgRamPtr, ecx
add ecx, 2
mov (ebp).IOMsgRam, ecx
add ecx, 2
mov (ebp).IORbdbufLen, ecx
```

```
add ecx, 2
mov (ebp).IORbdbufNum, ecx
add ecx, 2
mov (ebp).IORbtrControlAddr, ecx
```

```
add ecx, 2
mov (ebp).IOAfcControlAddr, ecx
add ecx, 2
mov (ebp).IOBitDetControlAddr, ecx
```

```
add ecx, 2
mov (ebp).IOAgeFirControlAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```
add ecx, 2
mov (ebp).IOCrkThrLowAddr, ecx
```

```

; base + 10h + 1ah
add     ecx, 2
mov     [ebp].IODaOffsetControlAddr, ecx
ddr = base + 10h + 1ch
add     ecx, 2
mov     [ebp].IOUnitIDAddr, ecx
10h + 1eh
add     ecx, 2

movzx   ecx, [ebx].MLIDIOPortsAndLengths
mov     al, 0bh
cmp     ecx, 100h
je      SetPort
dec     al
cmp     ecx, 140h
je      SetPort
dec     al
cmp     ecx, 180h
je      SetPort
dec     al
cmp     ecx, 1c0h
je      SetPort
dec     al
cmp     ecx, 200h
je      SetPort
dec     al
cmp     ecx, 240h
je      SetPort
dec     al
cmp     ecx, 280h
je      SetPort
dec     al
cmp     ecx, 2c0h
je      SetPort
dec     al
cmp     ecx, 300h
je      SetPort
dec     al
cmp     ecx, 340h
je      SetPort
dec     al
cmp     ecx, 380h
je      SetPort
dec     al

SetPort:
mov     dx, 279h
out     dx, al

; Lets Reset the adapter.
;
push     eax
mov     edx, [ebp].IOControl
mov     eax, CNTL_MRESET
out     dx, ax

mov     eax, [ebp].TimerTag
push     eax
push     2
call    DelayMyself
add     esp, (2 * 4)

mov     edx, [ebp].IOControl
mov     eax, 0
out     dx, ax

; IO DA Offset Control A
;
; Unit ID Addr = base +
;
; AL = 0bh
; I/O port 100h?
; yes
; AL = 0ah
; I/O port 140h?
; yes
; AL = 9
; I/O port 180h?
; yes
; AL = 8
; I/O port 1c0h?
; yes
; AL = 7
; I/O port 200h?
; yes
; AL = 6
; I/O port 240h?
; yes
; AL = 5
; I/O port 280h?
; yes
; AL = 4
; I/O port 2c0h?
; yes
; AL = 3
; I/O port 300h?
; yes
; AL = 2
; I/O port 340h?
; yes
; AL = 1
; I/O port 380h?
; yes
; AL = 0

; Set Base I/O port

; Make sure that we can set spare output
;
mov     edx, [ebp].IOControl
mov     eax, CNTL_SOUTPUT
out     dx, ax

mov     edx, [ebp].IOStatus
in      ax, dx
test    eax, STAT_SOUTPUT
mov     eax, offset MsgIOSetFailed
je      DriverInitErrorReturn

; Make sure that we can clear spare output
;
mov     edx, [ebp].IOControl
mov     eax, 0
out     dx, ax

mov     edx, [ebp].IOStatus
in      ax, dx
test    eax, STAT_SOUTPUT
mov     eax, offset MsgIOClearFailed
jne     DriverInitErrorReturn

; If no node override, default it.
;
cmp     dword ptr [ebx].MLIDNodeAddress, -1
jnz     short NodeIsSet
mov     [ebx].MLIDNodeAddress+0, 00h
mov     [ebx].MLIDNodeAddress+1, 80h
mov     [ebx].MLIDNodeAddress+2, 0aeh
mov     [ebx].MLIDNodeAddress+3, 00h
mov     [ebx].MLIDNodeAddress+4, 00h
mov     [ebx].MLIDNodeAddress+5, 01h

NodeIsSet:
; .....
; Download the MIPS code to the adapter.
; .....
;
mov     edx, [ebp].IOControl
mov     eax, CNTL_AUTO_INC
out     dx, ax

mov     edx, [ebp].IOMsgRamPtr
xor     eax, eax
out     dx, ax

mov     edx, [ebp].IOMsgRamPtr
xor     eax, eax
out     dx, ax

mov     ecx, MipsCodeSize
mov     edx, [ebp].IOMsgRam
lea     esi, MipsCode
cld
lodsw
out     dx, ax

CopyToAdapterLoop:
; Get Mips Word
; Send it to adapter

```



```

je      OpenScreenExit
mov     ScreenRtag, eax

push    offset DPCScreen
push    eax
push    offset ScreenName
call    OpenScreen
lea     esp, [esp + (3 * 4)]
or      eax, eax
jne     OpenScreenExit

push    offset NLNName
push    0
push    DriverModuleHandle
call    GetNLNNames
lea     esp, [esp + (3 * 4)]

push    0
push    offset Day
push    offset Month
push    offset Year
push    0
push    offset MinorVer
push    offset MajorVer
push    DriverModuleHandle
call    GetNLNVersionInfo
lea     esp, [esp + (8 * 4)]

push    Year
push    Day
push    Month
push    MinorVer
push    MajorVer
push    offset NLNName
push    offset DebugScreenHeader
push    DPCScreen
call    OutputToScreen
lea     esp, [esp + (8 * 4)]

OpenScreenExit:
; .....
; Initialize RxControl and Filter entries.
; .....
; .....

lea     edi, [ebp].RxControl
mov     ecx, MAX_CHAN
xor     eax, eax
mov     edx, RBD_NOT_USED

InitRxControlLoop:
mov     [edi].RxChannel, edx
mov     [edi].RXESR, eax
add     edi, size RX_CNTL
dec     ecx
jne     InitRxControlLoop

lea     edi, [ebp].Filter
mov     ecx, MAX_ADDIR

InitFilterLoop:
mov     [edi].FilterChannel, edx
mov     [edi].FilterTotalCount, eax
mov     [edi].FilterSeqCount, eax
mov     [edi].FilterSeqNum, eax

```

```

add     edi, size FilterStruct
dec     ecx
jne     InitFilterLoop
; .....
; Test Interrupts.
; .....
; Set ISR to test routine.
;
mov     [ebp].GotInterrupt, 0 ; Clear test flag.
mov     ecx, [ebp].ISRTag ; ECX = ISR resource tag.
push    0 ; No ExtraOIFlag offset.
push    0 ; Shareflag.
push    0 ; End of chain flag.
push    ecx ; ISR Resource tag.
lea     eax, TestDriverISR ; ISR Entry point.
push    eax ; EAX = Interrupt number.
movzx   eax, [ebx].MLIDInterrupt ; Interrupt Number.
push    eax ; Set interrupt.
call    SetHardwareInterrupt ; Restore stack.
lea     esp, [esp + (6 * 4)]
or      eax, eax ; Error setting interrupt?
lea     eax, BadISRMsg ; EAX -> Error Message.
jnz     DriverInitErrorReturn ; Exit if so.

mov     edx, [ebp].IOControl
mov     eax, [ebp].IOEnableValue
or      dx, ax

sti
mov     eax, [ebp].TimerTag
push    eax
push    18
call    DelayMyself
add     esp, (2 * 4)
cli

movzx   eax, [ebx].MLIDInterrupt
lea     edx, TestDriverISR
push    edx
push    eax
call    ClearHardwareInterrupt
lea     esp, [esp + (2 * 4)]

mov     eax, [ebp].IOEnableValue
out     dx, ax

lea     eax, NoInterruptMsg ; Error message.
cmp     [ebp].GotInterrupt, 0 ; Did we get it?
je      DriverInitErrorReturn ; Jump if not.
; .....
; EBX -> Frame Data Space(Config Table).
; EBP -> Adapter Data Space.
; .....
; Let MSM Set Hardware Interrupt.
; .....
call    MSMSetHardwareInterrupt

```



```
mov [ebp].AgentRemoveRoutine, 0
```

DriverShutdownAdapter:

```
pushfd
cli
mov edx, [ebp].IOControl
xor eax, eax
out dx, ax

mov edx, [ebp].IOStatus
in ax, dx

or ecx, ecx
jne DriverShutdownExit

mov edx, [ebp].IOControl
mov eax, CNTRL_MRESET
out dx, ax

mov eax, DPCScreen
or eax, eax
je DriverShutdownScreenClosed

push eax
call CloseScreen
lea esp, [esp + (1 * 4)]
mov DPCScreen, 0
```

DriverShutdownScreenClosed:

```
mov eax, [ebp].ProtocolBindID
or eax, eax
je DriverShutdownExit

push eax
call UnRegisterEventNotification
add esp, (1 * 4)

mov eax, [ebp].ProtocolUnbindID
or eax, eax
je DriverShutdownExit

push eax
call UnRegisterEventNotification
add esp, (1 * 4)
```

DriverShutdownExit:

```
popfd
xor eax, eax
ret

; Good Return code.
```

```
DriverShutdown endp
subttl -- DriverRemove --
page
```

```
; BEGIN_MANUAL_ENTRY( DriverRemove, DPC/API/REMOVE )
```

```
; Name: DriverRemove
```

```
; Description: This routine call the MSM to return our resources.
```

```
; On Entry: EAX N/A
; EBX N/A
; ECX N/A
; EDX N/A
; EBP N/A
; ESI N/A
```

```
N/A
```

```
Note: Interrupts are in any state.
```

```
On Return: EAX Destroyed
EBX Preserved
ECX Destroyed
EDX Destroyed
EBP Preserved
ESI Preserved
EDI Preserved
```

```
Flags:
```

```
Note: Interrupts preserved.
```

```
Remarks: This routine is called by the OS at unload.
It is called at process time.
```

```
See Also: MSM\MSMDriverRemove
```

```
END_MANUAL_ENTRY
```

```
DriverRemove proc
```

```
CPush
mov eax, DriverModuleHandle
call MSMDriverRemove
CPop
ret
```

```
DriverRemove endp
```

```
OSCODE ends
```

```
end
```

```
/* interface between Helios DPCNE and Hughes DPCPE */
```

```
extern "C" {
#include <nwssemaph.h>
#include "sys_win.hhi"
#include "dpcutils.h"
#include "dbsinwin.h"
#undef VIRTUAL
#include "dpcagent.h"
#include <assert.h>

int PD_ESR(ECB*);
void DloHangup(void);
void DPCPDTerminate(void);
void DPCPDBackground(void);
void DPCFileMain(void* arg); // thread
}

#include "sfswatch.h"
#include "sfqxview.h"
#include "sfxparser.h"

unsigned long GetTickCount(void) {
    return clock() * 1000 / CLOCKS_PER_SEC;
}

extern int DloState;
extern LONG DloPxmmitCount;
extern LONG DloPmaxBufferSize;
extern LONG DloRcvCount;
extern LONG DloConn;

int DloGetCurrentState(void) {
    if 1
        return (DloState == DLOS_CONN && DloConn == DLO_CONN_PACKAGE) ? DLOS_CONN : DL
    OS_IDLE;
    #else
        return DloState;
    #endif
}

int DloPortEmpty(void) {
    if 1
        return DloPxmmitCount == 0;
    #else
        return DloAndCommEmpty();
    #endif
}

int DloPortOpen(void) {
    return AiOPortHandle != (-1);
}

int DloGetStatus(tDloStatus* pStatus) {
    if (pStatus == 0)
        return (-1);
    if (!DloPortOpen())
        return (-1);
    pStatus->iState = DloGetCurrentState();
    pStatus->iXmitBytesBuffered = DloPxmmitCount;
    pStatus->iXmitBufferSize = DloPmaxBufferSize;
    pStatus->iRcvBytesBuffered = DloRcvCount;
    pStatus->iRcvBufferSize = DLOBUFSIZE;
    return 0;
}

int DloGetBufSize(void) {
```

```
return DloPmaxBufferSize - DloPxmmitCount;
}

DWORD DloExtendInactivityTimer(long) {
}

void DloHangup(void) {
}

void DloDispatch(void) {
}

/*
 * Returns whether the adapter can gain access to the passed group ID
 * The group ID includes a version number.
 */
long DLLAPI CDBCheckGroupID(CdbCfg_t *cfg)
{
    if(!find_pacau(cfg->groupid, cfg->ver) != NULL)
        return(CAS_IMPLICIT);
    if(!find_dacau(cfg->groupid, cfg->ver) != NULL)
        return(CAS_AUTHENTICATED);
    if(!find_ecau(cfg->groupid, cfg->ver) != NULL)
        return(CAS_EXPLICIT);
    return(CAS_ERROR);
}

/*
 * Returns a version number which increments when there have been
 * ANY changes to the adapter's conditional access.
 */
long DLLAPI CDBCheckCACHange(void)
{
    return CDBVersion;
}

struct PID {
    PID() { DPCFilePID = GetThreadID(); }
    ~PID() { DPCFilePID = 0; }
};

struct Semaphore {
    LONG handle;
    Semaphore(long initial = 0) { handle = OpenLocalSemaphore(initial); }
    ~Semaphore() { if (handle) CloseLocalSemaphore(handle); }
    void Signal(void) { SignalLocalSemaphore(handle); }
    LONG Wait(int milliseconds = (-1)) { return TimedWaitOnLocalSemaphore(handle,
        (LONG)milliseconds); }
    LONG value(void) { return ExamineLocalSemaphore(handle); }
    LONG operator --(void);
};

inline LONG Semaphore::operator --(void) {
    LONG v = value();
    if (v == 0)
        return v;
    WaitOnLocalSemaphore(handle);
    return v - 1;
}

Semaphore* DPCPDSemaphore;
SfxDispatcher* pDispatcher;
QUEUEVIEWER* pQueueViewer;
ECBQueue DPCPDQueue;
```

```

int PD_ESR(ECB* ecb) (
    Enqueue_IntsDisabled(&DPCPDQueue, ecb);
    return 0;
)

long BicddSignText(char* p_string,
    unsigned long size,
    char* p_sign) (
    return DIOSignText(p_string, size, p_sign);
)

long BicddGetSN(char* p_serial_num) (
    DIOGetSN(p_serial_num);
    return 0;
)

long BicddOpenChannel(BICDD_CHANNEL_CONFIG* channel_config) (
    if (channel_config->num_addresses != 1)
        return 1;
    DPCPDSemaphore = new Semaphore();
    if (DPCPDSemaphore->handle == 0) (
        delete DPCPDSemaphore;
        DPCPDSemaphore = 0;
        return 2;
    )
    DPCPDQueue.semaphore = DPCPDSemaphore->handle;
    // there is actually an overflow here IRT channel being a short!
    long ret = DIOOpenChannel(channel_config->address[0],
        PD_ESR,
        (LONG*)&channel_config->channel);
    if (ret) (
        delete DPCPDSemaphore;
        DPCPDSemaphore = 0;
        DPCPDQueue.semaphore = 0;
    )
    return ret;
)

long BicddCloseChannel(unsigned long channel) (
    long ret = DIOCloseChannel(channel);
    if (ret)
        return ret;
    delete DPCPDSemaphore;
    DPCPDSemaphore = 0;
    while (DPCPDQueue.head) (
        ECB* ecb = Dequeue(&DPCPDQueue);
        CLSLReturnRcvECB(ecb);
    )
    return 0;
)

.....
*
*      ELEMENTS SECTION
*      ( Elements Table support )
*
* .....
CDBelement_t Elements(MAXELEMENTS);
static find_element_by_mac(MAC_addr_t mac)
{
    int k;
    int ret = -1;
    for(k = 0; k < MAXELEMENTS; k++) (
        if(Elements[k].in_use == 'Y' &&
            memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) (
            ret = k;
            break;
        )
    )
    return ret;
}

static add_element(unsigned long channel, ID id, unsigned char ver,
    MAC_addr_t mac, char pack_feed)
{
    int k, ret = CAS_OK;
    if(find_element_by_mac(mac) != -1)
        return(CAS_DUPLICATE_ADDR);
    for(k = 0; k < MAXELEMENTS; k++)
        if(Elements[k].in_use != 'Y')
            break;
    if(k == MAXELEMENTS)
        ret = CAS_ERROR;
    else (
        Elements[k].channel = channel;
        Elements[k].e_ver = ver;
        memcpy(&Elements[k].e_id, &id, sizeof(id));
        memcpy(&Elements[k].e_mac, &mac, sizeof(mac));
        Elements[k].in_use = 'Y';
        Elements[k].packfeed = pack_feed;
    )
    return ret;
}

static find_element_id(ID id, unsigned char ver)
{
    int k;
    int ret = -1;
    for(k = 0; k < MAXELEMENTS; k++) (
        if(Elements[k].in_use == 'Y' &&
            memcmp(&Elements[k].e_id, &id, sizeof(id)) == 0 &&
            Elements[k].e_ver == ver) (
            ret = k;
            break;
        )
    )
    return ret;
}

static del_element_by_mac(MAC_addr_t mac)
{
    int k, ret = CAS_ERROR;
    for(k = 0; k < MAXELEMENTS; k++) (
        if(Elements[k].in_use == 'Y' &&
            memcmp(&Elements[k].e_mac, &mac, sizeof(mac)) == 0) (
            ret = CAS_OK;
            Elements[k].in_use = 'N';
            break;
        )
    )
    return ret;
}

```

```

.....
* Add / Delete Package Delivery Address
*/
/*
* Allows an application to request reseption of a single additional DPC MAC
* address. Caller supplies the address's elementID and version number and the
* element's group ID and version number. CDB looks up the group key and
* element key for the address and attempts to add the address via a
* driver call
*/
long BicddAddPKGAddr(CdbCfg_t* cfg) {
    char e_id_txt[7];
    MUXpacau_t* pacau;
    MUXdacau_t* dacau;

    make_element_id((BYTE*)&cfg->elementid, e_id_txt);
    MACbuildAddr(e_id_txt, MAC_PKG, cfg->ver, &cfg->mac);

    dacau = find_dacau(cfg->groupid, cfg->ver);
    pacau = dacau ? (MUXpacau_t*)dacau : find_pacau(cfg->groupid, cfg->ver);
    if (pacau == NULL)
        return CAS_ERROR;
    if (add_element(cfg->channel, cfg->elementid, cfg->ver, cfg->mac, 'P'))
        return CAS_ERROR;
    if (DIOAddGroupAddress(cfg->channel,
        (BYTE*)&cfg->mac,
        (BYTE*)&pacau->g_key) ==
        ESUCCESS)
        return CAS_OK;
    del_element_by_mac(cfg->mac);
    return CAS_ERROR;
}

/*
* For use by package delivery. Allows an application to request
* reception of a for-sale package ( a package from an explicit group).
* Package delivery passes address to be received (including the version number)
* plus the group key to be used to receive the package. This group key was
* received via explicit request transaction with the NOC.
* CDB creates the corresponding element key and calls WBicddAddress.
*/
long BicddAddExpAddr(CdbCfg_t* cfg) {
    if (find_eacu(cfg->groupid, cfg->ver) == 0)
        return CAS_ERROR;

    char e_id_txt[7];
    make_element_id((BYTE*)&cfg->elementid, e_id_txt);
    MACbuildAddr(e_id_txt, MAC_PKG, cfg->ver, &cfg->mac);
    if (add_element(cfg->channel, cfg->elementid, cfg->ver, cfg->mac, 'P'))
        return CAS_ERROR;
    if (DIOAddGroupAddress(cfg->channel,
        (BYTE*)&cfg->mac,
        (BYTE*)&cfg->expl_g_key) == ESUCCESS)
        return CAS_OK;
    del_element_by_mac(cfg->mac);
    return CAS_ERROR;
}

/*
* The name of the registry/ini key values accessed in this module
static char* PREGKEY_DeleteOnDelivery = "DeleteOnDelivery";
static char* PREGKEY_CooperativeLoading = "CooperativeLoading";
static char* PREGKEY_RebuildOnStartup = "RebuildOnStartup";
*/

```


Thu Jul 17 14:46:12 1997

dpcpd.cpp

```
)
pDispatcher = new SfxDispatcher();
if (!pDispatcher) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct SfxDispatcher");
    goto cleanup;
}
pQueueViewer = new QUEUEVIEWER(PDI_UpdatedDisplay);
if (!pQueueViewer) {
    DBS_SEND_ERROR(DBS_FATAL, "Could not construct QUEUEVIEWER");
    goto cleanup;
}
if (DPCGetProfileInt(PROF_PACKAGEDELIVERY, PREGKEY_Reconcile, 1))
    fsm.ReconcileWith(frd);
cdb.RebuildDB();
while (!ExitingFlag) {
    pDispatcher->Run();
    DPCPDBackground();
}
pDispatcher->Stop(5000);

cleanup:
delete pQueueViewer;
delete pDispatcher;
delete pDispatcherLro;
```

```
/* Our header file */
```

```
/* "fix" conflicting types */
#define AllocateResourceTag __AllocateResourceTag__
#include <advanced.h>
#include <AllocateResourceTag
#include <nwbitops.h>

#define milliclock() (GetHighResolutionTimer() / 10)

#define activityTimer ECB_DriverWorkspace.Dws_i32val

/* various flags that control the filter */
#define FILTER_DATA_ON_RST
#define WIDEN_TCP_WINDOW
#define TCP_ACK_LATENCY /* 10 */
/* if used at all, define *only* 1 of the following */
#define DPCinetMaxQueuedBytes /* 4096 */
#define DPCinetMaxQueuedPackets 64
/* if defined(DPCinetMaxQueuedBytes) && defined(DPCinetMaxQueuedPackets)
error Only 1 of DPCinetMaxQueuedBytes and DPCinetMaxQueuedPackets allowed
#endif
```

```
/* various flags that control the tunnel */
#define TUNNEL_ONLY_TCP
```

```
#define IP_VERS(x) ((BYTE*)(x)[0] >> 4)
#define IP_HDR_LEN(x) ((BYTE*)(x)[0] & 0x0f)
#define IP_TOS(x) ((BYTE*)(x)[1])
#define IP_TOT_LEN(x) ((WORD*)(x)[1])
#define IP_FLAG_FRAG(x) ((WORD*)(x)[3])
#define IP_PROTO(x) ((BYTE*)(x)[9])
#define IP_CSUM(x) ((WORD*)(x)[5])
#define IP_SRC_ADDR(x) ((LONG*)(x)[3])
#define IP_DST_ADDR(x) ((LONG*)(x)[4])
```

```
#define IPPROTO_IPENCAP 0x04
```

```
#define UDP_SRC_PORT(x) ((WORD*)(x)[0])
#define UDP_DST_PORT(x) ((WORD*)(x)[1])

#define TCP_SRC_PORT(x) ((WORD*)(x)[0])
#define TCP_DST_PORT(x) ((WORD*)(x)[1])
#define TCP_ACKNUM(x) ((LONG*)(x)[2])
#define TCP_CODE(x) ((BYTE*)(x)[13])
#define TCP_WINDOW(x) ((WORD*)(x)[7])
#define TCP_CSUM(x) ((WORD*)(x)[8])
```

```
#define TCP_FIN 0x01
#define TCP_SYN 0x02
#define TCP_RST 0x04
#define TCP_PSH 0x08
#define TCP_ACK 0x10
#define TCP_URG 0x20
```

```
ECBQueue TxQ;
```

```
ECBQueue NewQ;
```

```
struct ResourceTagStructure* TxChainRTag = 0;
struct ResourceTagStructure* TxChainRTag = 0;
LONG TxChainID;
struct ResourceTagStructure* RxChainRTag = 0;
struct ResourceTagStructure* RxChainRTag = 0;
LONG RxChainID;
LONG DPC_IP_Address = 0;
static BYTE ConnectIonMask[65536 / 8];
```

```
#ifndef __GNUC__
#define inline
#endif /* __GNUC__ */

/* ECB Manipulation */

static inline void ReleaseECB(ECB* ecb) {
    if (DIOStats) {
        #ifdef DPCinetMaxQueuedBytes
            if ((DIOStats->QDepth -= ecb->ECB_DataLength) < 0)
                DIOStats->QDepth = 0;
        #endif
        #ifdef DPCinetMaxQueuedPackets
            --DIOStats->QDepth;
        #endif
    }
    --TxECBRTag->RTResourceCount;
    CLSUFastSendComplete(ecb);
    #ifdef LOG_ECB_ACTIVITY
        FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
            "TINET Release(%08lx)\n", ecb));
    #endif /* LOG_ECB_ACTIVITY */
}

inline void Enqueue_IntsDisabled(ECBQueue* q, ECB* ecb) {
    ecb->ECB_NextLink = 0;
    if (q->tail)
        q->tail->ECB_NextLink = ecb;
    ecb->ECB_PreviousLink = q->tail;
    q->tail = ecb;
    if (q->head == 0)
        q->head = ecb;
    SignalLocalSemaphore(q->semaphore);
}

void Enqueue(ECBQueue* q, ECB* ecb) {
    _disable();
    Enqueue_IntsDisabled(q, ecb);
    _enable();
}

ECB* Dequeue(ECBQueue* q) {
    ECB* ecb;
    _disable();
    ecb = q->head;
    if (ecb == 0) {
        _enable();
        return 0;
    }
    q->head = ecb->ECB_NextLink;
    if (q->head == 0)
        q->tail = 0;
    else
        q->head->ECB_PreviousLink = 0;
    _enable();
    ecb->ECB_NextLink = ecb->ECB_PreviousLink = 0;
    return ecb;
}
```

```

void Remove(ECBQueue* q, ECB* ecb) (
_disable();
if (ecb->ECB_NextLink)
    ecb->ECB_NextLink->ECB_PreviousLink = ecb->ECB_PreviousLink;
else
    q->tail = ecb->ECB_PreviousLink;
if (ecb->ECB_PreviousLink)
    ecb->ECB_PreviousLink->ECB_NextLink = ecb->ECB_NextLink;
else
    q->head = ecb->ECB_NextLink;
_enable();
ecb->ECB_NextLink = ecb->ECB_PreviousLink = 0;
)

LONG InetQueuePacket(ECB* ecb, LONG board, void* chainID) (
    board = board;
    chainID = chainID;
    /* only handle IP packets */
    if (!(LONG*)ecb->ECB_ProtocolID != 0 ||
        *(WORD*)(ecb->ECB_ProtocolID + 4) != htons(0x0800))
        return 1;

#ifdef LOG_ECB_ACTIVITY
    if (LogECBHandle) {
        int TGID = SetThreadGroupID(DPC_TGID);
        LogMsg(LogClientHandle, LogECBHandle, FALSE,
            "TINET Enqueue(%08lx)\n", ecb);
        SetThreadGroupID(TGID);
    }
#endif /* LOG_ECB_ACTIVITY */
    Enqueue(&NewQ, ecb);
    return 0;
)

LONG InetControl(void) (
    return 0xfffff91;
)

void ClearConnection(WORD port) (
    if (ScanBits(ConnectionMask, port, port+2) == port) (
        BitClear(ConnectionMask, port);
        --DIOStats->TxOKMultipleCollisions;
    )
)

int AllocateConnection(WORD port) (
    if (ScanBits(ConnectionMask, port, port+2) != port) (
        /* see if there is a connection left */
        if (DIOStats->TxOKMultipleCollisions < DPCMaxConnections) (
            /* allocate the new connection */
            BitSet(ConnectionMask, port);
            ++DIOStats->TxOKMultipleCollisions;
            return 1;
        )
        return 0;
    )
    return 1;
)

LONG ConnectionLimiter(ECB* ecb, LONG board, void* chainID) (

```

```

)
BYTE* IPHeader = ecb->ECB_Fragment[0].FragmentAddress;
BYTE* TCPHeader = 0;
board = board;
chainID = chainID;
/* not used */
/* not used */

/* only handle IP packets */
if (!(LONG*)ecb->ECB_ProtocolID != 0 ||
    *(WORD*)(ecb->ECB_ProtocolID + 4) != htons(0x0800))
    return 1;

/* double check stats, hopefully upper layer is kosher, but */
if (DIOStats == 0) {
    releaseECB;
    --RxCBRTag->RTRSourceCount;
    CLSUFastSendComplete(ecb);
    return 0;
}

/* only check TCP packets to our interface */
if (IP_PROTO(IPHeader) != IPPROTO_TCP ||
    IP_DST_ADDR(IPHeader) != DPC_IP_Address)
    return 1;

TCPHeader = IPHeader + IP_HD_LEN(IPHeader) * 4;
if (ecb->ECB_Fragment[0].FragmentLength < ((TCPHeader + 20) - IPHeader))
    return 1;

if (TCP_CODE(TCPHeader) & (TCP_FIN|TCP_RST)) (
    /* release the connection */
    ClearConnection(ntohs(TCP_DST_PORT(TCPHeader)));
)
else if (TCP_CODE(TCPHeader) & TCP_SYN) (
    /* allocate the connection */
    if (!AllocateConnection(ntohs(TCP_DST_PORT(TCPHeader))))
        goto releaseECB;
)
return 1;
)

/* IP Manipulation */
#if 0
char *chksum (BYTE *buf, unsigned cnt)
(
    static unsigned char crc_bytes[2];
    BYTE rbl;
    WORD rax, rcx;
    int redx;
    BYTE *rdssi;

    crc_bytes[0] = crc_bytes[1] = 0;

    rcx = cnt;
    rdssi = buf;
    rbl = rcx;
    rcx = rcx >> 1;
    redx = 0;
    if (rcx != 0)
    (
        while (rcx--)
        (
            rax = *((WORD *)rdssi);
            rdssi += 2;

```

```

        if (redx & 0xffff0000)
            redx++;
        redx &= 0x0000ffff;
        redx += rax;
    }
    if (redx &= 0x0000ffff)
    {
        redx &= 0x0000ffff;
        redx++;
    }
    if (rbl & 1)
    {
        rax = 0;
        rax = *rdss;
        redx += rax;
        if (redx &= 0x0000ffff)
            redx++;
    }
    redx = -redx;
    crc_bytes[0] = redx & 0xff;
    crc_bytes[1] = (redx >> 8) & 0xff;
    return (char *)crc_bytes;
};

#endif

#ifdef __GNUC__
/*
 * This is a version of ip_compute_csum() optimized for IP headers, which
 * always checksum on 4 octet boundaries.
 * This version is constructed from various places in the linux and Hughes
 * sources.
 */
static inline unsigned short ip_fold_lcomp_csum(unsigned long sum) {
    unsigned short csun;
    __asm__ ("movl %w1, %w0\n\t"
             "shrl $16, %1\n\t"
             "addw %w1, %w0\n\t"
             "adcw $0, %w0\n\t"
             "notw %w0"
             : "=a" (csun)
             : "b" (sum));
    return csun;
}

```

```

static inline unsigned short ip_fast_csum(unsigned short *buff, int wlen) {
    unsigned long sum = 0;

    if (wlen) {
        unsigned long eax;
        /* Suggested speedup:
        1:
        movl (%esi), %ebx
        lea (%esi+4), %esi
        adcl %ebx, %eax
        decl %ecx
        jnz 1b
        adcl $0, %eax
        movl %eax, %ebx
        shrl $16, %eax
        addw %ebx, %eax
        adcl $0, %eax
        */
    }
}

```

```

        xorl $0xffff, %eax
        */
        __asm__ ("cld\n\t"
                 "lods\n\t"
                 "adcl %3, %0\n\t"
                 "loop 1b\n\t"
                 "adcl $0, %0\n\t"
                 : "=r" (sum), "=S" (buff), "=c" (wlen), "=a" (eax)
                 : "0" (sum), "1" (buff), "2" (wlen));
    }
    return ip_fold_lcomp_csum(sum);
}

#define chksum(b, l) ip_fast_csum(b, (l) / 4)

static inline unsigned short ip_adjust_csum(unsigned short oldcsum,
                                             unsigned short oldval,
                                             unsigned short newval) {
    unsigned long sum = ((unsigned short)-oldcsum);
    sum += ((unsigned short)-oldval);
    sum += newval;
    return ip_fold_lcomp_csum(sum);
}

#endif /* __GNUC__ */

static int DummyFrame (FRAG_DESC* frag) {
    frag = frag;
    return 0;
} /* not used */

int (*DPCDropFrame) (FRAG_DESC* frag) = DummyFrame;

void FilterQueue(void* arg) {
    ECB* ecb;
    ECB* rover;
    BYTE* IP;
    BYTE* TCP;
    int excess;
    arg = arg; /* not used */

    RenameThread(GetThreadID(), "DPCAgent Filter");

    for (;;) {
        if (ExitingFlag)
            return;
        TimedWaitOnLocalSemaphore(NewQ.semaphore, 1000);
        if (!NewQ.head)
            continue;

        ecb = Dequeue(&NewQ);
        ecb->activityTimer = milliclock();
        IP = ecb->ECB_Fragment[0].FragmentAddress;

        if (DIOWait == 0) {
            releaseECB;
            DPCDropFrame((FRAG_DESC*)&ecb->ECB_FragmentCount);
            --TxECBRTag->RTResourceCount;
            CLSLFastSendComplete(ecb);
        }
        ifdef LOG_ECB_ACTIVITY
            FastLogMsg(LogECBHandle, (LogClientHandle, LogECBHandle, TRUE,
            "TINET Release(%08lx)\n", ecb));
        }
    }
}

```

```
#endif /* LOG_ECB_ACTIVITY */
continue;
}
```

```
/* always send a fragmented or routed packet */
if (ecb->ECB_Fragment[0].FragmentLength < 20 ||
    IP_FLAG_FRAG(IP) & htons(0x3fff) ||
    IP_SRC_ADDR(IP) != DPC_IP_Address) {
```

```
    enqueueTxQ;
```

```
#ifdef DPCInetMaxQueuedBytes
```

```
    if (DIOSStats->QDepth > DPCInetMaxQueuedBytes) {
        ++DIOSStats->RetryTxCount;
        goto releaseECB;
    }
```

```
    DIOSStats->QDepth += ecb->ECB_DataLength;
```

```
#endif
```

```
#ifdef DPCInetMaxQueuedPackets
```

```
    if (DIOSStats->QDepth > DPCInetMaxQueuedPackets) {
        ++DIOSStats->RetryTxCount;
        goto releaseECB;
    }
```

```
    ++DIOSStats->QDepth;
```

```
#endif
```

```
    Enqueue(&TxQ, ecb);
```

```
    continue;
```

```
}
```

```
excess = ecb->ECB_Fragment[0].FragmentLength - IP_HD_LEN(IP) * 4;
```

```
if (excess > 0) {
```

```
    TCP = IP + IP_HD_LEN(IP) * 4;
```

```
}
```

```
else {
```

```
    TCP = ecb->ECB_Fragment[1].FragmentAddress + (-excess);
```

```
    excess += ecb->ECB_Fragment[1].FragmentLength;
```

```
}
```

```
if (IP_PROTO(IP) == IPPROTO_UDP)
```

```
    goto filterUDP;
```

```
if (IP_PROTO(IP) != IPPROTO_TCP)
```

```
    goto enqueueTxQ;
```

```
filterTCP:
```

```
if (excess < 20)
```

```
    goto enqueueTxQ;
```

```
if (TCP_CODE(TCP) & TCP_SYN) {
```

```
    if (!AllocateConnection(ntohs(TCP_SRC_PORT(TCP))))
```

```
        goto releaseECB;
```

```
/* scan for duplicate in TxQ */
```

```
for (rover = TxQ.head; rover = rover->ECB_NextLink) {
```

```
    BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
```

```
    BYTE* roverTCP;
```

```
    excess = (rover->ECB_Fragment[0].FragmentLength -
```

```
        IP_HD_LEN(roverIP) * 4);
```

```
    if (excess > 0) {
```

```
        roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
```

```
    }
```

```
    else {
```

```
        roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
```

```
        excess += rover->ECB_Fragment[1].FragmentLength;
```

```
    }
```

```
    if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
```

```
        excess >= 20 &&
```

```
        (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
```

```
        IP_PROTO(roverIP) == IPPROTO_TCP &&
```

```
TCP_CODE(roverTCP) == TCP_CODE(TCP) &&
    IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
    TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
    IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
    TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP)) {
    ++DIOSStats->TxOKSingleCollision;
```

```
    goto releaseECB;
```

```
}
```

```
    goto enqueueTxQ;
```

```
}
```

```
#ifdef FILTER_DATA_ON_RST
```

```
    if (TCP_CODE(TCP) & TCP_RST) {
```

```
        /* scan for data in TxQ */
```

```
for (rover = TxQ; rover; rover = rover->ECB_NextLink) {
```

```
    BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
```

```
    BYTE* roverTCP;
```

```
    excess = (rover->ECB_Fragment[0].FragmentLength -
```

```
        IP_HD_LEN(roverIP) * 4);
```

```
    if (excess > 0) {
```

```
        roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
```

```
    }
```

```
    else {
```

```
        roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
```

```
        excess += rover->ECB_Fragment[1].FragmentLength;
```

```
    }
```

```
    if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
```

```
        excess >= 20 &&
```

```
        (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
```

```
        IP_PROTO(roverIP) == IPPROTO_TCP &&
```

```
        TCP_DST_PORT(roverTCP) == IP_DST_PORT(TCP) &&
```

```
        IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
```

```
        TCP_SRC_PORT(roverTCP) == IP_SRC_PORT(TCP) &&
```

```
        TCP_CODE(roverTCP) != TCP_CODE(TCP)) {
```

```
            rover->activityTimer = 0; /* will get taken out shortly */
```

```
            ++DIOSStats->TxAbortCarrierSense;
```

```
        }
```

```
/* fallthru */
```

```
}
```

```
#endif
```

```
if (TCP_CODE(TCP) & (TCP_RST|TCP_FIN)) {
```

```
    ClearConnection(ntohs(TCP_SRC_PORT(TCP)));
```

```
    goto enqueueTxQ;
```

```
}
```

```
if (TCP_CODE(TCP) & TCP_ACK) {
```

```
    #ifdef WIDEN_TCP_WINDOW
```

```
        WORD oldwin = TCP_WINDOW(TCP);
```

```
        WORD newwin = ntohs(oldwin);
```

```
        if (newwin < 40000) {
```

```
            newwin += (newwin >> 1);
```

```
            newwin = htons(newwin);
```

```
            TCP_WINDOW(TCP) = newwin;
```

```
            TCP_CSUM(TCP) = ip_adjust_csum(TCP_CSUM(TCP),
```

```
                oldwin,
```

```
                newwin);
```

```
        }
```

```
#endif
```

```
if (TCP_CODE(TCP) & (TCP_URG|TCP_PSH))
```

```
    goto enqueueTxQ;
```

```
#ifdef TCP_ACK_LATENCY
```

```
    ecb->activityTimer = milliclock() + TCP_ACK_LATENCY;
```

```
#endif
```

```
/* scan for redundancy in TxQ */
```

```
for (rover = TxQ.head; rover = rover->ECB_NextLink) {
```

```
    if (TCP_CODE(TCP) & (TCP_URG|TCP_PSH))
```

```
        goto enqueueTxQ;
```

```
#ifdef TCP_ACK_LATENCY
```

```
    ecb->activityTimer = milliclock() + TCP_ACK_LATENCY;
```

```
#endif
```

```
/* scan for redundancy in TxQ */
```

```
for (rover = TxQ.head; rover = rover->ECB_NextLink) {
```

```

BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
BYTE* roverTCP;
excess = (rover->ECB_Fragment[0].FragmentLength -
IP_HD_LEN(roverIP) * 4);
if (excess > 0) {
    roverTCP = roverIP + IP_HD_LEN(roverIP) * 4;
}
else {
    roverTCP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
    excess += rover->ECB_Fragment[1].FragmentLength;
}
if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
    (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
    IP_PROTO(roverIP) == IPPROTO_TCP &&
    IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
    TCP_DST_PORT(roverTCP) == TCP_DST_PORT(TCP) &&
    IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
    TCP_SRC_PORT(roverTCP) == TCP_SRC_PORT(TCP) &&
    TCP_CODE(roverTCP) & TCP_ACK &&
    (htonl(TCP_ACKNUM(roverTCP)) + htons(TCP_WINDOW(roverTCP)) <
    htonl(TCP_ACKNUM(TCP)) + htons(TCP_WINDOW(TCP))) {
    /* move ACK information over to TxQ and release this packet */
    TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
    TCP_WINDOW(roverTCP),
    TCP_WINDOW(TCP));
    TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
    (WORD)TCP_ACKNUM(roverTCP),
    (WORD)TCP_ACKNUM(TCP));
    TCP_CSUM(roverTCP) = ip_adjust_csum(TCP_CSUM(roverTCP),
    TCP_ACKNUM(roverTCP)>>16,
    TCP_ACKNUM(TCP)>>16);
    TCP_ACKNUM(roverTCP) = TCP_ACKNUM(TCP);
    TCP_WINDOW(roverTCP) = TCP_WINDOW(TCP);
    ++DIOSStats->TxAbortExcessCollisions;
    goto releaseECB;
}
goto enqueueTxQ;
goto enqueueTxQ;

filterUDP:
{
    BYTE* UDP = TCP;
    BYTE* DNS;

    /* ECB contents determined by inspection, there are safer methods */
    if (excess < 8)
        goto enqueueTxQ;

    /* filter DNS only */
    if (UDP_DST_PORT(UDP) != htons(53))
        goto enqueueTxQ;

    excess -= 8;
    DNS = (excess > 0) ? (UDP + 8) : ecb->ECB_Fragment[1].FragmentAddress;

    for (rover = TxQ.head; rover = rover->ECB_NextLink) {
        BYTE* roverIP = rover->ECB_Fragment[0].FragmentAddress;
        BYTE* roverUDP;
        BYTE* roverDNS;
        excess = (rover->ECB_Fragment[0].FragmentLength -
        IP_HD_LEN(roverIP) * 4);
        if (excess > 0) {
            roverUDP = roverIP + IP_HD_LEN(roverIP) * 4;

```

```

}
else {
    roverUDP = rover->ECB_Fragment[1].FragmentAddress + (-excess);
    excess += rover->ECB_Fragment[1].FragmentLength;
}
if (rover->ECB_Fragment[0].FragmentLength >= 20 &&
    (excess >= 8 &&
    (IP_FLAG_FRAG(roverIP) & htons(0x3fff)) == 0 &&
    IP_PROTO(roverIP) == IPPROTO_UDP &&
    IP_DST_ADDR(roverIP) == IP_DST_ADDR(IP) &&
    UDP_DST_PORT(roverUDP) == UDP_DST_PORT(UDP) &&
    IP_SRC_ADDR(roverIP) == IP_SRC_ADDR(IP) &&
    UDP_SRC_PORT(roverUDP) == UDP_SRC_PORT(UDP) &&
    (roverDNS = (((excess - 8) > 0) ?
    (roverUDP + 8) :
    rover->ECB_Fragment[1].FragmentAddress)) &&
    * (LONG*)DNS == *(LONG*)roverDNS) {
    ++DIOSStats->TxAbortLateCollision;
    goto releaseECB;
}
goto enqueueTxQ;
}
}

/* SLIP, PPP, Modem Manipulation */
#define MAX_READ_BUF 128

int InetState = MODEM_IDLE;
static BYTE SlipEndPkt[1] = {END};

int WaitingLines = 0, NextWait = 0;
char WaitingBuffer[MAX_READ_BUF];
int WaitingIndex = 0;
LONG ConnectingTimeout = 0;
LONG ConnectingRedial = FALSE;

int BaudRate[] = {
    2400, /* 0 */
    3600, /* 1 */
    4800, /* 2 */
    7200, /* 3 */
    9600, /* 4 */
    19200, /* 5 */
    38400, /* 6 */
    57600, /* 7 */
    115200 /* 8 */
};

void InitLogin()
{
    int i;
    char *nextWait;
    WaitingLines = 0;
    if (DioCfg.auto_login)
    {
        WaitingIndex = 0;
        WaitingBuffer[WaitingIndex] = '\0';
        NextWait = 0;
    }
}

```

```

ConnectingTimeout = 0;
for (i = 0, nextWait = Dlocfg.wait_for_1; i < 9; i++, nextWait =
sizeof(Dlocfg.wait_for_1))

```

```

    if (*nextWait)
        waitingLines++;
}

```

```

static BYTE MTUBuffer[8192];

```

```

int SLIPSendRoutineOpt(FRAG_DESC* fragStruc)

```

```

{
    LONG count = 0;
    BYTE* output = MTUBuffer;

    *output++ = END;
    while (count < fragStruc->FragmentCount)
    {
        FRAGMENTSTRUCT* frag = fragStruc->FragmentDesc + count;
        BYTE* frame = (BYTE*)frag->FragmentAddress;
        LONG length = frag->FragmentLength;

        while (length-- > 0)
        {
            switch (*frame)
            {
                case END:
                    *output++ = ESC;
                    *output++ = ESC_END;
                    break;
                case ESC:
                    *output++ = ESC;
                    *output++ = ESC_ESC;
                    break;
                default:
                    *output++ = *frame;
                    break;
            }
            if (header == 0x80000000)
                header = (*frame & 0x0f) * 4;
            if (--header == 0)
                dataStart = output;
            ++frame;
        }
        ++count;
    }
}

```

```

if (output - MTUBuffer < 20 ||
    output - MTUBuffer > DloGetWriteBufferSize())
    return 0;
DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
DloSend(MTUBuffer, dataStart - MTUBuffer, DLO_INET_TIMEOUT);
DloSend(dataStart, output - dataStart, DLO_INET_TIMEOUT);
DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
return 1;
}

int (*DPTxFrame)(FRAG_DESC* fragStruc) = SLIPSendRoutineOpt;

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

int SLIPSendRoutineDebug(FRAG_DESC* fragStruc)

```

```

{
    LONG count = 0;
    BYTE* output = MTUBuffer;
    BYTE* dataStart = 0;
    LONG header = 0x80000000;

    while (count < fragStruc->FragmentCount)
    {
        /* separate HEADER from PAYLOAD */

```

```

FRAGMENTSTRUCT* frag = fragStruc->FragmentDesc + count;
BYTE* frame = (BYTE*)frag->FragmentAddress;
LONG length = frag->FragmentLength;

```

```

while (length-- > 0)
{
    switch (*frame)
    {
        case END:
            *output++ = ESC;
            *output++ = ESC_END;
            break;
        case ESC:
            *output++ = ESC;
            *output++ = ESC_ESC;
            break;
        default:
            *output++ = *frame;
            break;
    }
    if (header == 0x80000000)
        header = (*frame & 0x0f) * 4;
    if (--header == 0)
        dataStart = output;
    ++frame;
}
++count;
}

```

```

if (output - MTUBuffer < 20 ||
    output - MTUBuffer > DloGetWriteBufferSize())
    return 0;
DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
DloSend(MTUBuffer, dataStart - MTUBuffer, DLO_INET_TIMEOUT);
DloSend(dataStart, output - dataStart, DLO_INET_TIMEOUT);
DloSend(SlipEndPkt, 1, DLO_INET_TIMEOUT);
return 1;
}

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

/*
 * IPSendRoutine(ECB *tcb)
 *
 * Description:
 *
 * Input:   tcb
 *          Control Block
 *
 * Output:  nothing
 *
 * Returns: 0 if finished with ECB
 */
static BYTE IPHeader[IP_TUNNEL_SIZE] =
{
    0x45,
    0,
    0, 0,
}
/* version 4, length 5 */
/* tos */
/* length */

```

```

0, 0,
0, 0,
0x7f,
4,
);
#define IPHeaderIdent (*(WORD*)&IPHeader[4])

int
IPSendRoutine(ECB *ecb)
{
    FRAG_DESC* fragStruc = alloca(sizeof(LONG) + (sizeof(FRAGMENTSTRUCT) * (
        ECB->ECB_FragmentCount + 2)));
    WORD frame_size = ecb->ECB_DataLength;
    int options_collapsed = 1;
    LONG currFrag = 0;
    BYTE* ecbIPHeader = ecb->ECB_Fragment[0].FragmentAddress;

    /* Initialize the copy of the tcb fragStruc */
    memcpy(fragStruc,
        &ecb->ECB_FragmentCount,
        sizeof(LONG) + (sizeof(FRAGMENTSTRUCT) * ecb->ECB_FragmentCount));

    if (frame_size < DloCfg.mtu &&
        TUNNEL_ONLY_TCP
        /* UDP doesn't need tunnel header */
        (ecbIPHeader[9] != IPPROTO_TCP) ||
        /* either do "routed" packets */
        ((*LONG*)&ecbIPHeader[12] != DPC_IP_Address))
        goto skipFragger;

    memset(&fragStruc->FragmentDesc(fragStruc->FragmentCount),
        0,
        sizeof(FRAGMENTSTRUCT) * 2);

    frame_size += IP_TUNNEL_SIZE;

    /* fill IPHeader with tunnel data, including IP/gateway addresses,
     * and prepend to frag list.
     */
    *(WORD*)&IPHeader[2] = htons(frame_size);
    ++IPHeaderIdent;
    *(WORD*)&IPHeader[10] = 0; /* checksum, for now */
    *(LONG*)&IPHeader[12] = DloCfg.ip_address;
    *(LONG*)&IPHeader[16] = DloCfg.gateway_address;
    memmove(fragStruc->FragmentDesc + 1,
        fragStruc->FragmentDesc,
        sizeof(FRAGMENTSTRUCT) * fragStruc->FragmentCount);
    fragStruc->FragmentDesc[0].FragmentAddress = IPHeader;
    fragStruc->FragmentDesc[0].FragmentLength = IP_TUNNEL_SIZE;
    ++fragStruc->FragmentCount;
    ++currFrag;
    *(WORD*)&IPHeader[10] = chksum((WORD *)IPHeader,
        IP_TUNNEL_SIZE);

    while (frame_size > DloCfg.mtu)
    {
        /*
         * Shucks. Have to fragment the packet.
         * This algorithm is roughly per RFC791.
         */
        LONG OIHL = fragStruc->FragmentDesc[0].FragmentLength;
        BYTE OMF = IPHeader[6] & 0x20;
        LONG NPH (DloCfg.mtu - OIHL) & 0xffff;
        WORD TL = OIHL + NPH;

```

```

        IPHeader[6] |= 0x20; /* set More Fragments */
        *(WORD*)&IPHeader[2] = htons(TL);
        *(WORD*)&IPHeader[10] = 0; /* clear checksum */
        *(WORD*)&IPHeader[10] = chksum((WORD *)IPHeader, OIHL);

        /*
         * Now fake out the fragStruc to reflect TL.
         * Hang on to enough information to remove the TL less OIHL
         * later.
         */
        TL -= OIHL;
        frame_size -= TL;
        while (TL > 0 &&
            fragStruc->FragmentDesc[currFrag].FragmentLength <= TL)
        {
            TL -= fragStruc->FragmentDesc[currFrag].FragmentLength;
            ++currFrag;
        }
        if (TL > 0)
        {
            /*
             * This frag gets split into 2 pieces.
             */
            memmove(fragStruc->FragmentDesc + currFrag + 1,
                fragStruc->FragmentDesc + currFrag,
                sizeof(FRAGMENTSTRUCT) *
                    (fragStruc->FragmentCount - currFrag));
            ++fragStruc->FragmentCount;
            fragStruc->FragmentDesc[currFrag].FragmentLength = TL;
            ++currFrag;
            fragStruc->FragmentDesc[currFrag].FragmentLength -= TL;
            fragStruc->FragmentDesc[currFrag].FragmentAddress = ((ch
                ar*)fragStruc->FragmentDesc[currFrag].FragmentAddress) + TL;
        }
        TL = fragStruc->FragmentCount - currFrag + 1;
        fragStruc->FragmentCount = currFrag;

        if (DPCtxFrame(fragStruc) == 0)
            return 0;

        if (!options_collapsed) {
            LONG offset = 20;
            while (offset < OIHL && IPHeader[offset]) {
                if (IPHeader[offset] & 0x80) /* copy */
                    offset += IPHeader[offset + 1];
                else /* collapse */
                    LONG len = IPHeader[offset + 1];
                memcpy(IPHeader + offset,
                    IPHeader + offset + len,
                    OIHL - (offset + len));
                OIHL -= len;
            }
            offset = fragStruc->FragmentDesc[0].FragmentLength;
            fragStruc->FragmentDesc[0].FragmentLength = (OIHL + 3) &
                0x3c;
            memset(IPHeader + OIHL,
                0,
                fragStruc->FragmentDesc[0].FragmentLength - OIHL);
            IPHeader[0] = 0x40 | (fragStruc->FragmentDesc[0].Fragment
                length / 4);
            frame_size -= offset - fragStruc->FragmentDesc[0].Fragme
                ntLength;
            options_collapsed = 1;
        }
    }
}

```

```

/*
 * Adjust the frag list to "remove" the fragment just sent.
 */
memmove(fragStruc->FragmentDesc + 1,
        fragStruc->FragmentDesc + currFrag,
        sizeof(FRAGMENTSTRUCT) * TL);
fragStruc->FragmentCount = TL;
currFrag = 1;

/* compute new IPHeader values: fragment offset */
*(WORD*)(&IPHeader[6]) = htons((intohs(*(WORD*)(&IPHeader[6])) &
                                0x1fff)
                                + (NFB / 8));

IPHeader[6] |= OMF;
if (frame_size <= DloCfg.mtu)
{
    *(WORD*)(&IPHeader[2]) = htons(frame_size);
    *(WORD*)(&IPHeader[10]) = 0; /* Clear checksum */
    *(WORD*)(&IPHeader[10]) = chksum((WORD *)IPHeader,
                                     fragStruc->FragmentDesc
                                     [0].FragmentLength);
    break;
}

skipFragger:
/* send the remaining (possibly ALL) portion of the frame */
if (DPCTXFrame(fragStruc) == 0)
    return 0;

if (DIOStats)
{
    ++DIOStats->TotalTxPacketCount;
    if ((DIOStats->TotalTxOKByteCountLow += ecb->ECB_DataLength) <
        ecb->ECB_DataLength)
        ++DIOStats->TotalTxOKByteCountHigh; /* wrapped */
}
return 1;

static void EmptyESR(ECB* ecb) {
}

unsigned char RawEnvelope[14] = {
    0x00, 0x00, 0x00, 0x0c, 0x0a, 0x0b, 0x0c,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x08, 0x00,
};

ECB RawECB = {
    0,
    0,
    0,
    EmptyESR,
    -1,
    -1,
    0,
    -1,
    0,
    -1,
    0,
    -1,
    -1,
    (RawEnvelope, sizeof(RawEnvelope))
};

/* technically unsafe, but works */
FRAGMENTSTRUCT RawFragments[15] = {
    IPHeader, sizeof(IPHeader),
};

#define RawSendRoutineDebug RawSendRoutineOpt
int RawSendRoutineOpt (FRAG_DESC* fragStruc) {
    int i;
    for (i = 20000; --i > 0; ) {
        if (RawECB.ECB_Status == 0)
            goto rawSend;
        if (ExitingFlag)
            return 0;
        ThreadSwitchWithDelay();
    }
    return 0;
rawSend:
    i = fragStruc->FragmentCount;
    RawECB.ECB_FragmentCount = i + 1;
    memcpy(RawFragments,
           fragStruc->FragmentDesc,
           i * sizeof(FRAGMENTSTRUCT));
    RawECB.ECB_DataLength = sizeof(RawEnvelope);
    while (i > 0)
        RawECB.ECB_DataLength += RawFragments[--i].FragmentLength;
    CLSLSendPacket(&RawECB);
    return 1;
}

void DisplayWaitStatus(void)
{
    char statusStr[80];
    char *waitingStr;

    if (WaitingLines == 0)
        return;

    waitingStr = (char *)&DloCfg.wait_for_1[NextWait*30];
    NWSprintf(statusStr, MSG("Modem Status: Waiting for \"%s\"\\n", 557), wait
ingStr);
    UpdateModemStr(statusStr);

    void InetStateChange(int state) {
        if (DloCfg.out_protocol == OUT_NETWORK) {
            InetState = PROTOCOL_CONNECTED;
            return;
        }
        switch (state) {
            case DLOS_IDLE:
            case DLOS_DISC_1:
            case DLOS_DISC_2:
            case DLOS_DISC_3:
            case DLOS_DISC_4:
                InetState = MODEM_IDLE;
                if (ConnectingRedial) {
                    DloEndConn();
                    ConnectingRedial = FALSE;
                }
                break;
            case DLOS_CONN:
                InetState = MODEM_CONNECTED;
                InitLogin();
                if (InetAsleep)
                    ResumeThread(DPCInetPID);
                break;
        }
    }
}

```

```

default:
break;
)

/*****
* FUNCTION: Convert Internet address
*
* DESCRIPTION: converts a character string containing the Internet address
* into a form that BIC DD understands.
* e.g. 139.85.124.06 (8B.55.7C.06) into 067C558B0000
*****/
void convert_address(char *lpszIpAddress)
{
    char *p;
    int i = 0;
    char tmp[20], tmp1[10];
    tmp[0] = 0;
    while((p=strchr(lpszIpAddress, (int)'.')) != NULL)
    {
        i = atoi(p+1);
        sprintf(tmp1, MSG("%02X", 477), i);
        strcat(tmp, tmp1);
        *p = 0;
    }
    i = atoi(lpszIpAddress);
    sprintf(tmp1, MSG("%02X", 478), i);
    strcat(tmp, tmp1);
    CStrCpy(lpszIpAddress, tmp);
}

MACAddr_t      HIAddr;
LONG            InetChannel;

void make_hi_key(chunk *key)
{
    int i;
    LONG sn;
    BYTE serialNum[9];
    BYTE serialNumPacked[3];
    BYTE x;

    DIOGetSN(serialNum);
    sn = atol(serialNum);
    sprintf(serialNum, MSG("%061x", 480), sn);

    pack_mac_addr(serialNumPacked, 3, serialNum, 6);
    x = serialNumPacked[0];
    serialNumPacked[0] = serialNumPacked[2];
    serialNumPacked[2] = x;

    key->b[0] = serialNumPacked[0] ^ 0xff;
    key->b[1] = serialNumPacked[1] ^ 0xff;
    key->b[2] = serialNumPacked[2] ^ 0xff;
    for(i = 3; i < 8; i++)
        key->b[i] = 0x00 ^ 0xff;

    MACbuildAddr(serialNum, MAC_HI, 0, &HIAddr);
}

void InetChangeProtocol(void)
{
    switch (DioCfg.out_protocol) {
    case OUT_PPP:
        DPCTxFram = DebugFlag ? PPPEndRoutineDebug : PPPEndRoutineOpt;
        break;
    case OUT_NETWORK:
        void (*ControlEntryPoint)(void) = 0;
        struct DriverConfigurationStructure* dvrCfg = 0;
        if (CLSLGetMLIDControlEntry(DioCfg.net_interface,
                                   &ControlEntryPoint))
        {
            goto skipDriver;
        }
        dvrCfg = (struct DriverConfigurationStructure*)
            CommandMlid(DioCfg.net_interface, 0, (LONG)ControlEntryP
            oint);
        memcpy(RawEnvelope + 6, dvrCfg->DNodeAddress, 6);

        skipDriver:
        RawECB.ECB_BoardNumber = DioCfg.net_interface;
        memcpy(RawEnvelope, DioCfg.net_addr, 6);
        DPCTxFram = DebugFlag ? RawSendRoutineDebug : RawSendRoutineOpt;
        break;
    case OUT_SLIP:
        DPCTxFram = DebugFlag ? SLIPSendRoutineDebug : SLIPSendRoutineOpt;
        break;
    }
    InetStateChange(DLOS_DISC_4);
    DioEndConn();

    int ProcessLogin(void)
    {
        BYTE value;
        char *sendStr, *waitStr;
        char sendBuf[40];
        LONG nextTimeout;

        /* No use trying if we aren't even connected */
        /* Get out if we're done */
        if (WaitingLines == 0 || DioCfg.auto_login == FALSE)
        {
            return(TRUE);
        }
        if (!DioConnected())
            return(FALSE);
        /* Timeout if we've waited too long for this wait */
        if (ConnectingTimeout == 0)
        {
            ConnectingTimeout = GetCurrentTime() + DioCfg.wait_timeout * 1
            8;
        }
        if (GetCurrentTime() > ConnectingTimeout)
    }

```

```

    if (ConnectingRedial == FALSE)
    {
        /* First timeout. Send return and try again. */
        ConnectingRedial = TRUE;
        InitLogin();
        DloSend(MSG("\r", 181), 1, DLO_INET_TIMEOUT);
        return(FALSE);
    }
    DloEndConn();
    return(FALSE);
}

DisplayWaitStatus();

while (DloReceive(&value, 1) != 0)
{
    if (DebugFlag)
        putchar(value);
    if (value != '\r' && value != '\n')
    {
        WaitingBuffer[WaitingIndex++] = value;
        WaitingBuffer[WaitingIndex] = 0;
        if (WaitingIndex > (MAX_READ_BUF-1))
            WaitingIndex = 0;
    }

    waitStr = (char *)&DloCfg.wait_for_1[NextWait * 30];
    if (!strchr(WaitingBuffer, waitStr) != NULL)
    {
        sendStr = (char *)&DloCfg.send_1[NextWait * 30];
        NWSprintf(sendBuf, MSG("%s\r", 558), sendStr);
        DloSend(sendBuf, CStrlen(sendBuf), DLO_INET_TIMEOUT);
        NextWait++;
        WaitingIndex = 0;
        WaitingBuffer[WaitingIndex] = '\0';
        WaitingLines--;
        if (WaitingLines == 0)
        {
            DloUpdateModemStr();
            return(TRUE);
        }
        DisplayWaitStatus();
        nextTimeout = DloCfg.wait_timeout_1 + NextWait;
        ConnectingTimeout = GetCurrentTime() +
            (nextTimeout) ? (nextTimeout * 18) : (5
*18));
        return(FALSE);
    }
    else if (value == '\r')
    {
        WaitingIndex = 0;
        WaitingBuffer[WaitingIndex] = '\0';
    }
}

return(FALSE);
}

int ConnectProtocol(void)
{
    int ccode;

    if (DloCfg.out_protocol == OUT_SLIP)
    {
        delay(1000);
        return 1;
        /* time to "settle" */
    }
}

```

```

void TinetcProtocolBind(LONG __parameter) (
    struct EventProtocolBindStruct* epbs =
    (struct EventProtocolBindStruct*)__parameter;
    if (epbs->boardNumber == DIOBoard &&
        epbs->protocolNumber == 1/*PROTOCOL_ID_TCPIP*/) (
        extern LONG DPCNextRegistrationCheck;
        DPCGetIPAddress(&DPC_IP_Address);
        DPCNextRegistrationCheck = 0;
    )
)

/*.....*/
*
* InetMain(void *parm)
*
* Description:
*   Main thread for Turbo Internet handling.
*
* Input:
*   parm
*
* Output:
*   nothing
*
* Returns:
*   nothing
*
*.....*/
- ignored

void InetMain(void *parm)
{
    time_t nextStartConn = 0;
    LONG removedCount = (LONG)(-1);
    long millidelay = 0;
    LONG protocolBindHandle =
        RegisterForEvent(EVENT_PROTOCOL_BIND, TinetcProtocolBind, 0);

    parm = parm; /* unused */

    NewQ_semaphore = OpenLocalSemaphore(0);
    TxQ_semaphore = OpenLocalSemaphore(0);
    BeginThread(FilterQueue, 0, 0, 0);

    TxChainRTag = AllocateResourceTag(NLMHandle,
        MSG("Turbo Inet TxPreScan Chain", 476))
        LSLTxPreScanStackSignature);
    TxECBRTag = AllocateResourceTag(NLMHandle,
        MSG("Turbo Inet Transmit Packets", 619),
        ECBSignature);
    RxChainRTag = AllocateResourceTag(NLMHandle,
        "Turbo Inet RxPreScan Chain",
        LSLPreScanStackSignature);
    RxECBRTag = AllocateResourceTag(NLMHandle,
        MSG("Turbo Inet Receive Packets", 619),

```

```

DPCGetIPAddr(&DPC_IP_Address);
ECBSignature);

if (Dlocfg.out_protocol == OUT_NETWORK)
    InetState = PROTOCOL_CONNECTED;

mainloop:
while (!ExitingFlag)
{
    InetAsleep = TRUE;
    if (millidelay > 55)
        delay(millidelay);
    else if (millidelay > 0)
        ThreadSwitchWithDelay/*LowPriority*/();
    else
        ThreadSwitch();
    InetAsleep = FALSE;

    while (DIOBoard && removedCount != DIORemovedCount)
    {
        BYTE address[8];
        BYTE szBicBCDAddress[20];
        struct DriverStatsStructure* stats = 0;
        LONG ip_address = ntohl(Dlocfg.ip_address);

        removedCount = DIORemovedCount;
        /* Enable internet reception */

        /* Yuk. We'll change this later to get rid these extra s
        NWSprintf(szBicBCDAddress, MSG("%d.%d.%d.%d", 620),
            (ip_address >> 24) & 0xff,
            (ip_address >> 16) & 0xff,
            (ip_address >> 8) & 0xff,
            (ip_address) & 0xff);

        convert_address(szBicBCDAddress);
        if (!pack_mac_addr(address, 6,
            szBicBCDAddress, CStrLen(szBicBCDAddr
            ess)))

        address
        \n", xxxx)); */
        millidelay = 500;
        break;
    }

    /* Sending an esr address of -1 tells MLID to handle rec
    if (DIOOpenChannel(address,
        (int (*)())0xffffffff,
        &InetChannel))
    {
        millidelay = 500;
        removedCount = (LONG)(-1);
        break;
    }

    if (ExitingFlag)
        break;
    DIOAddHIAddr(InetChannel, (BYTE *)&HIAddr);
    DPCGetMLIDStats(&stats);
    DIOStats->TxOrMultipleCollisions = 0;
    if (CULSRegisterPreScanTxChain(TxChainRTag,
        DIOBoard,
        3, /* next to last */
        &TxChainID,
    }
}

InetQueuePacket,
InetControl,
TxECBRTag))

millidelay = 500;
removedCount = (LONG)(-1);
break;
}

if (CULSRegisterPreScanRxChain(RxChainRTag,
    DIOBoard,
    3, /* next to last */
    &RxChainID,
    ConnectionLimiter,
    InetControl,
    RxECBRTag))

millidelay = 500;
removedCount = (LONG)(-1);
break;
}

if (Dlocfg.out_protocol == OUT_PPP &&
    InetState == PROTOCOL_CONNECTED)
{
    PPPBackground();
}

if (TxQ.head == 0 &&
    (InetState <= MODEM_CONNECTING ||
    InetState >= PROTOCOL_CONNECTED))
{
    TimedWaitOnLocalSemaphore(TxQ.semaphore, 200);
    millidelay = 0;
    continue;
}

switch (InetState)
{
case MODEM_CONNECTED:
    if (!ProcessLogin())
    {
        millidelay = 500;
        break;
    }
    InetState = LOGIN_CONNECTED;
    /* fallthru */
case LOGIN_CONNECTED:
    if (!ConnectProtocol())
    {
        DIOEndConn();
        millidelay = 15 * 1000;
        break;
    }
    InetState = PROTOCOL_CONNECTED;
    millidelay = 1;
    break;
case PROTOCOL_CONNECTED:
    LONG count = 0;
    if (Dlocfg.out_protocol != OUT_NETWORK &&
        AIOWriteStatus(AIOPortHandle, &count, 0))
    {
        millidelay = 200;
        break;
    }
}

```

```

if (count == 0)
{
    LONG milliclock = milliclock();
    ECB* ecb;
    ecb = TxQ.head;
    millidelay = 100;
    while (ecb->activityTimer > milliclock)
    {
        LONG diff = ecb->activityTimer - milliclock;

        if (diff < millidelay)
            millidelay = diff;
        if ((ecb = ecb->ECB_NextLink) == 0)
            goto mainloop;
    }
    Remove(&TxQ, ecb);
    if (ecb->activityTimer < milliclock() - 60000) {
        if (ecb->activityTimer)
            ++DIOStats->TXAbortExDeferral;
    }
    else
        IPSendRoutine(ecb);
    ReleaseECB(ecb);
    if (DIOCfg.out_protocol != OUT_NETWORK)
        AIOWriteStatus(AIOPortHandle, &count, 0);
}
millidelay = (count *
10 * /* Tx bits with framing */
1000 / /* milliseconds */
BaudRate(DIOCfg.tinet_baud_index));
break;
}
case MODEM_IDLE:
    if (nextStartConn < time(0))
    {
        InitLogin();
        DIOStartConn(DLO_INET_TIMEOUT);
        InetState = MODEM_CONNECTING;
        nextStartConn = time(0) + 30;
    }
    /* fallthru */
default:
    millidelay = 10 * 1000;
    break;
}

DIOCloseChannel(InetChannel);

CLSLDeRegisterPreScanRxChain(RxChainID);
CLSLDeRegisterPreScanTxChain(TxChainID);
while (TxQ.head)
    ReleaseECB(Dequeue(&TxQ));
while (NewQ.head)
    ReleaseECB(Dequeue(&NewQ));
CloseLocalSemaphore(TxQ.semaphore); TxQ.semaphore = 0;
CloseLocalSemaphore(NewQ.semaphore); NewQ.semaphore = 0;
UnregisterForEvent(protocolBindHandle);

DPCinetPID = 0;
return;
}

```

ock;